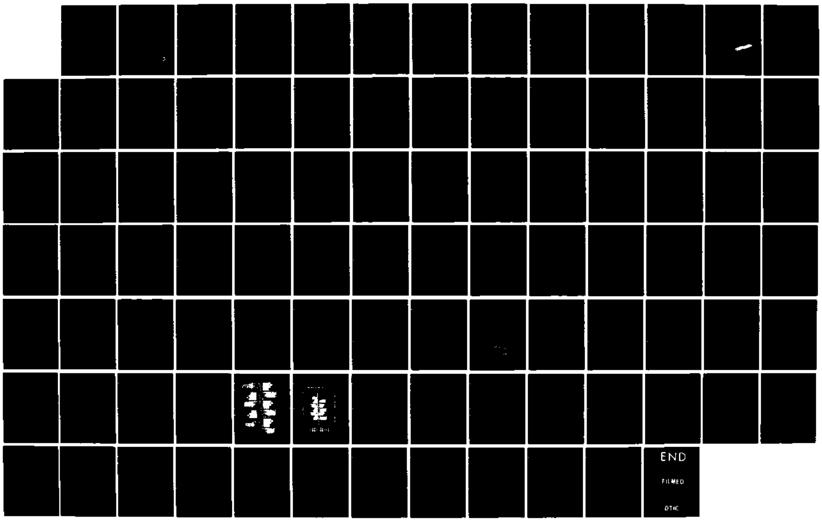
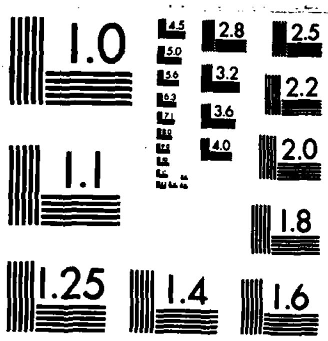
AD-A154 678 NEW DEVELOPMENTS ON THE CORE FUNCTION FOR EFFICIENT IMPLEMENTATION OF THE..(U) BOEING COMPUTER SERVICES CO TUKWILA WA R E ALTSCHUL ET AL. FEB 85 N00014-84-C-0130 1/1

UNCLASSIFIED F/G 12/1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0

(2)

NEW DEVELOPMENTS ON THE CORE FUNCTION FOR EFFICIENT IMPLEMENTATION OF THE DIFFICULT RESIDUE NUMBER SYSTEM OPERATIONS

Final Report
November 1983 through February 1985
Manuscript Completed: April 1985

Prepared by Roberto E. Altschul, Dale D. Miller[1], John N. Polky[2], James R. King[3], and David A. Uvelli[4]

Boeing Computer Services Division
The Boeing Company
565 Andover Park West
Mail Stop 9C-01
Tukwila, Washington 98188

Prepared for
OFFICE OF NAVAL RESEARCH
Statistics and Probability Branch
Arlington, VA. 22217
Under Contract No. N00014-84-C-0130

1. BAE Automated Systems Inc.
2. Sigma Research Inc.
3. University of Washington
4. Seattle Silicon Technology Inc.

Abstract. This paper develops new properties of the core function on a *residue number system* (RNS) which allows efficient implementation of the operations of comparison, overflow detection, sign determination, parity determination, scaling, and general division. Previously these operations have been considered difficult to implement in high speed hardware and have not taken advantage of the parallel structure of a *residue class architecture*. In 1977, Akushskii, Burcev and Pak introduced the core function and presented algorithms for these difficult operations. While these algorithms were superior to previous techniques, the evaluation of the core function required, in general, an iterative, complex procedure. Moreover, while these techniques were theoretically attractive, they were unable to construct a methodology for determining a suitable core function for a realistic moduli set. In the present work, a new method for evaluating the core function is introduced. This method utilizes a redundant modulus and its computational complexity is equivalent to that of the first iteration of the method of Akushskii et al. While this new method requires additional hardware to carry on this redundant modulus calculation, it provides more information and allows more flexibility than the previous method. New and much more efficient algorithms for the difficult RNS operations are developed. In addition, new structural properties of the core function are developed, and the optimality of the function is characterized. The selection of an optimal core function for any moduli set is cast as an integer programming problem. Finally, to ensure its applicability to real time signal processing hardware, the feasibility of implementing the core and other residue functions in VLSI circuits was investigated. It was determined that a core calculation can be implemented on a single chip in one 50 nsec clock cycle.

—A—

TABLE OF CONTENTS

I. INTRODUCTION

The Residue Number System (RNS) offers the potential for very high speed integer arithmetic for signal processing applications. However, due to the relative difficulty of performing such operations as sign detection, magnitude comparison, and parity checking for residue encoded numbers, it has been impractical to embed data depedent logical branching in the overall computational flow. Since many signal processing problems require the reduction of large volumes of data to a few, relatively simple decision paths, the RNS has had an insignificant impact on commercial or military signal processing systems. The primary goal of this project was to investigate new RNS methods to satisfy the nonlinear signal processing requirements of the Navy's Ocean Surveillance Signal Processing Program. A secondary goal of this effort was to investigate and evaluate recent Soviet developments in the use of the RNS for high speed computation. Initial investigations of the Soviet work yielded some promise of a solution to the nonlinear signal processing problem. In this paper the Soviet work is described and new developments presented which include new algorithms permitting hardware implementation of the nonlinear operations. This solution is superior to any previous RNS techniques and is competitive with a binary implementation in terms of computational latency, throughput, and hardware complexity.

A considerable body of Soviet work exists in the open literature, dating back to the mid 1960's. This includes well over fifty papers and two recent books, many of which are referenced in Miller et al [3]. A number of patents have been issued for the device implementation of many of these concepts, and special purpose RNS processors have been built. The principal investigators include I. Akushskii, V. Amerbaev, V. Burcev, I. Pak, and D. Yuditskii. Their results have both paralleled and diverged from Western work.

In the late 1960's, hardware architectures for complex residue systems were developed, utilizing an isomorphism from the complex RNS on the moduli set $(p_1 + iq_1, \ldots, p_n + iq_n)$ to the real RNS on the moduli set $(p_1^2 + q_1^2, \ldots, p_n^2 + q_n^2)$. Much of this early work concentrated on the use of redundant residue systems for error detection and correction. In the 1970's, subjects considered included the use of non-coprime moduli, quadratic residue systems, magnitude comparison, parity detection, RNS representations of rational numbers, fractional multiplication and division, rounding, and the use of optical techniques for RNS processing.

A recurrent theme in much of the work from the mid 1960's to the present has been the study of positional characteristics of residue encoded numbers. A positional characteristic of a number is a characteristic easily determined from a positional radix representation. These efforts were directed at defining a function from the RNS to the integers which could be easily evaluated and would permit efficient sign detection, comparison, and parity determination. In 1970 Amerbaev studied the use of threshold networks for determining positional characteristics, leading to a 1973 patent with Akushskii and others for its hardware implementation. During the 1970's, several techniques were proposed which offered detection of overflow under addition and other positional information when using specialized moduli sets.

In 1977, Akushskii, Burcev, and Pak introduced the notion of the core of a residue number [1], [2]. The core function provides an easily implemented and efficient technique for performing the traditionally difficult residue operations: sign detection, magnitude comparison, scaling, parity determination, overflow detection, and extension of base.

This paper reviews the concept and properties of the core, describes algorithms for performing the difficult residue operations, presents new techniques for the efficient evaluation of the core, and characterizes the problem of selecting an appropriate core function for a given moduli set. For example, comparison of two numbers in a residue system with n moduli can be performed in 3 clock periods using $n+5$ integrated circuits and can be pipelined to achieve a 20 MHz data rate.

Similar improvement over existing methods can be obtained for the other difficult residue operations..

Section II defines the core function and presents a number of its properties. Section III discusses the computation of the core. For a properly selected core function, the evaluation of the core at most integers within the RNS range is easily computed from its residue representation. However, in some cases this calculation yields an ambiguous result, and a more complicated algorithm is required to evaluate the core. In Section IV a new algorithm for core calculation is presented which utilizes a redundant modulus to avoid these difficulties. Another difficulty for the practical utilization of the core has been the selection of certain fixed coefficients which determine a particular core function. Section V shows how this can be recast as an integer programming problem which can be solved by standard tools to obtain an optimal core.

Section VI presents algorithms for magnitude comparison, and Section VII gives algorithms for general division. Two algorithms are presented in each section: a general algorithm for an arbitrary core function and an efficient algorithm for a suitably linear core. In Section VIII, the VLSI implementation of the core function and other RNS operations is discussed. The goal of this investigation was to ascertain the feasibility of implementing several basic residue operations on a single chip. Given the current status of VLSI design tools, the most efficient approach to this feasibility investigation is to perform complete design and layout of prototype VLSI circuits. This section describes coding considerations, logic reductions, and the VLSI design steps. The results of this work affirm the feasibility of implementing such algorithms as a core evaluation on a single chip at high speed and throughput.

The work of Akushskii et al. has continued, focusing on the use of the core function to perform floating point arithmetic using the RNS. In this setting, an integer a in the residue system actually represents the product of the proper fraction a/M with a power of 2, where M is the product of the moduli. Algorithms for addition, multiplication, division, and binary shifting have been developed, and a patent for a general purpose floating point arithmetic unit based on these concepts has appeared. These algorithms are described in the Appendix.

II. DEFINITION AND PROPERTIES OF THE CORE FUNCTION

Let $m_1, \ldots, m_n$ be the relatively prime moduli of a residue number system with product M. For any integer a, $0 \leq a < M$, there exists a unique n-tuple $(a_i) = (a_1, \ldots, a_n)$ with $0 \leq a_i < m_i$ for which $a \equiv a_i \pmod{m_i}$. In this case we write $a = (a_i)$. If b is an arbitrary integer satisfying $b \equiv a_i \pmod{m_i}$, we say b is a representative of (a_i) and write $b \equiv (a_i) \pmod{M}$. A residue encoding (a_i) will always be assumed to lie in the internal $[0, M)$. For any integers a and m, $|a|_m$ denotes the least non-negative residue of a modulo m, and if $(a,m) = 1$, $|1/a|_m$ denotes the multiplicative inverse of a modulo m.

The n-tuple $(a_1, \ldots, a_n)$ determines a unique integer a, $0 \leq a < M$. Nevertheless, unless decoded, the n-tuple conveys no positional information, i.e., magnitude comparisons are not possible from simple direct analysis of the n-tuples. It is of interest to find some means of attaining such information in an economic and computationally feasible way.

For a given modulus, say m_1, an integer is uniquely determined by the quotient $[a/m_1]$ and the residue $|a|_{m_1} = a_1$. Furthermore the information given by the pairs of (quotient, remainder) are sufficient for magnitude comparisons. If a and b are any two integers, then $a < b$ if and only if $[a/m_1] < [b/m_1]$ or, $[a/m_1] = [b/m_1]$ and $|a|_{m_1} < |b|_{m_1}$.

Thus if both $[a/m_i]$ and (a_i) are represented in the RNS, comparisons are possible. However, this approach is not practical since it leads to calculations on domains of just one order of magnitude smaller than that of the RNS, viz., for $a \in [0,M)$ the quotients $[a/m_i]$ take values in $[0,M/m_i)$. An alternative is to consider a simple function of the set of quotients $[a/m_i]$. The simplest form for such a function is an integer linear combination of the quotients. This is the form of the core function as defined by Akushskii et al.

Definition. Let $w_1,\ldots,w_n$ be fixed but arbitrary integers, not all zero. The core $C(a)$ of an arbitrary integer a is defined as

$$C(a) = \Sigma\, w_i [a/m_i], \tag{1}$$

where $[\cdot]$ denotes the greatest integer function.

The core coefficients w_i are fixed for the moduli set and do not depend on the integer a. The selection of the w_i is important for efficiency of the core calculation, cf. Sections III and IV. It is desirable that the w_i are chosen so that the range of the core function on $[0,M)$ is small, i.e., on the order of the individual moduli of the system. It is not obvious from the definition that $C(a)$ is easily computable from (a_i), since the definition would require that (a_i) first be decoded to obtain a, then n integer divisions be performed, and finally an inner product of length n taken. It will be shown later that the core can be obtained directly from the (a_i) as the residue of an inner product.

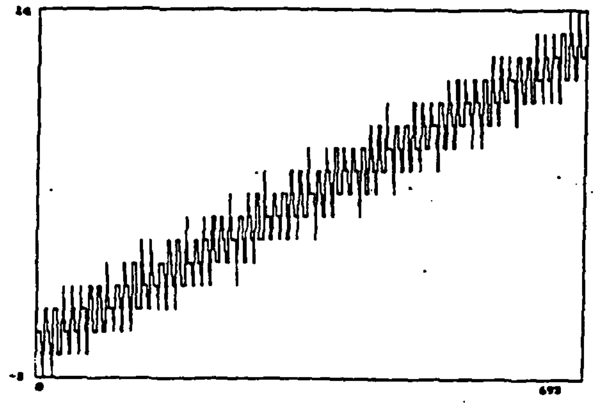
Note also from the definition that $C(a)$ is a step function. As a increases, $C(a)$ remains constant until a equals a multiple of any of the moduli, at which a step jump occurs. Note next that if all the w_i are positive, $C(a)$ is a non-decreasing function and hence contains direct positional information; however, $C(M)$ (and hence the range of the core function) will be quite large, and evaluation of the core function becomes computationally intractable. For example, if each $w_i = 1$, $C(M) = \Sigma\, w_i \hat{m}_i = \Sigma\, \hat{m}_i$, where $\hat{m}_i = M/m_i$. Thus it will be required that both positive and negative core coefficients are used, resulting in a core function which is no longer monotonic.

Akushskii et al. considered the core a "positional characteristic" of a residue encoded number. For computational reasons, they required selection of the w_i's so that $C(0)$ $(=0)$ and $C(M)$ are near the minimum and maximum cores on $[0,M)$, respectively. Consequently, the cores they considered tended to be nearly non-decreasing with step function graphs having some negative steps but generally following the straight line from *(0,0)* to *(M, C(M))*. Such cores will thus contain global positional information: $a << b$ if and only if $C(a) << C(b)$. Locally, however, such a core does not directly distinguish magnitudes. Thus in selecting the coefficients w_i for a core function, the tradeoff between compactness of range and monotonicity must be balanced.

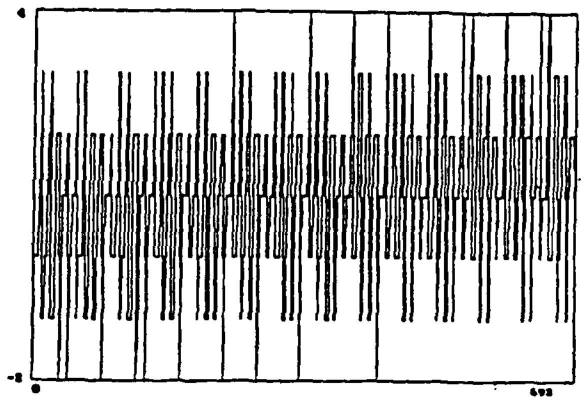
We will demonstrate a modified algorithm for computing cores which does not require the approximate monotonicity of the core function. Thus highly nonlinear, "wild" core functions which may have very small ranges can be considered. In Section IV it is shown that such wild core functions can be used for parity determination, extension of base, and scaling. However, for sign detection and magnitude comparison, a global linearity of the core is still required.

Figure 1 illustrates the graphs of four cores. The first graph is taken from an example in [2] having moduli set {7,9,11} and core coefficients {-1,-1,3}. The core is approximately monotonic, taking its *minimum near 0 and its maximum near* $M = 693$. Figure 1(b) illustrates a core for the same moduli set and core coefficients {1,2,-4}. For this moduli set, these coefficients give rise to the core of minimal range. In Figures 1 (c), (d), cores are shown for the more practical moduli set {23,25,27,29,31}. The core coefficients for (c) are {-3, -2, 4, -1, 3} and for (*d*) are {-1,8,-2,-4,-2}.

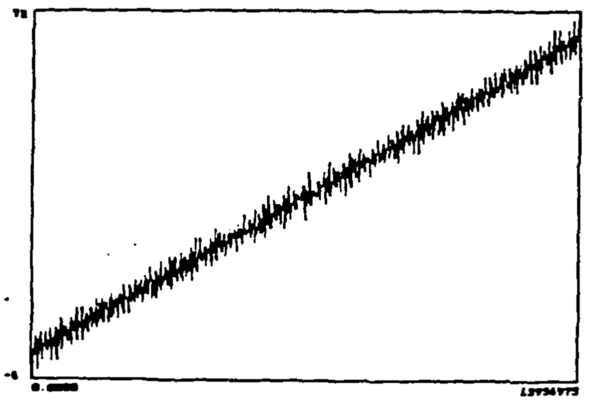
In this section, several properties of the core are derived and used to give efficient algorithms for the "difficult" residue operations. Since these algorithms require calculation of the core, the elegance of the algorithms cannot be fully appreciated until core calculation is discussed in Sections III and IV. Most of the theorems in this section are due to Akushskii et al., although the proofs have been considerably simplified and several errors corrected.



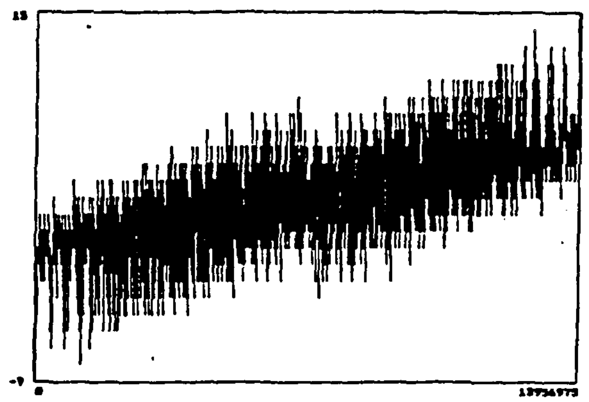
(a) moduli set = {7, 9, 11}
core weights = {-1, -1, 3}



(b) moduli set = {7, 9, 11}
core weights = {1, 2, 4}



(c) moduli set = {23, 25, 27, 29, 31}
core weights = {-3, -2, 4, -1, 3}



(d) moduli set = {23, 25, 27, 29, 31}
core weights = {-1, 8, -2, -4, -2}

Figure 1. Graphs of four core functions illustrate the trade between compactness of range and linearity.

e first theorem gives a formula for an integer a in terms of its core and its residue representation. thus provides a decoding algorithm which is similar to the Chinese remainder theorem with the vantage that no modulo M reductions are required.

ieorem 1 (Akushkii et al.). If a is any integer (not necessarily restricted to $[0,M)$), and $C(M) \neq 0$, en

$$a = \frac{M \cdot C(a) + \Sigma \alpha_i (\hat{m}_i w_i)}{C(M)}.$$

oof. Since

$$\left[\frac{a}{m_i} \right] = \frac{a - \alpha_i}{m_i},$$

) implies that

$$M \cdot C(a) = \Sigma w_i M (a - \alpha_i)/m_i$$

$$= a \Sigma w_i \hat{m}_i - \Sigma w_i \alpha_i \hat{m}_i.$$

he result follows by solving for a.

he nonlinearity of the core arises from the truncation of the greatest integer function. The next ieor n provides a measure of how close the core function is to being an additive homomorphism. his theor vides the basis for many of the ensuing results.

heorem 2 (Akushskii et al.). If $a \equiv (\alpha_i)$ (mod M) and $b \equiv (\beta_i)$ (mod M) are integers (not necessarily estricted to $[0,M)$), then

$$C(a + b) = C(a) + C(b) + \Sigma w_i \varepsilon_i,$$

here

$$\varepsilon_i = \left[\frac{\alpha_i + \beta_i}{m_i} \right] = 0 \text{ or } 1.$$

itical core is found. The lift stage then reverses the steps of the first stage, by removing the critical pect of the core. *This stage assumes core functions satisfy some degree of separability.*

t α be an integer with critical core. In the descent stage, a modulus is discarded and the core of α is mputed relative to the reduced moduli set, using a new set of coefficients w_i, pre-selected for the duced moduli set. If this core is critical, another modulus is discarded and the step repeated. The escent is terminated when a non-critical core is found. The descent is guaranteed to terminate at or efore the stage at which the reduced moduli set has only two moduli, since w_1 and w_2 can be chosen that no critical cores are present (e.g., $w_1 = 1$ and $w_2 = 1$, or $w_1 = 0$ and $w_2 = 1$).

he method of lifting appends iteratively the previously discarded moduli and determines the true ore at each stage. Assume the core is not critical for the residue encoded integer $(a_1,\ldots, a_i)$ relative the moduli set $m_1,\ldots,m_i$. With Theorem 9, this known core is used to extend this integer relative to e new basis element m_{i+1}. The resulting residual β must satisfy

$$\beta + k \cdot (m_1 \cdot \cdot \cdot m_i) \equiv a_{i+1} (mod\ m_{i+1}) \tag{5}$$

r some integer $k \in [0, m_{i+1})$. If the interval $[0, m_1 \cdots m_{i+1})$ is partitioned into m_{i+1} consecutive tervals containing $m_1 \cdots m_{i+1}$ integers each, k determines the number of the interval containing $_1,\ldots, a_{i+1})$. Thus the true core of $(a_1,\ldots, a_{i+1})$ relative to $m_1,\ldots, m_{i+1}$ can be determined if the core nction is m_{i+1} - separable. If the core function is relatively linear the process is reduced to a simple omparison: if k is less than a predetermined constant K then the true core is that which is close to ero, otherwise the true core is that which is close to the core at the product $m_1 \ldots m_{i+1}$. In the articular case where the constant K is equal to $[m_{i+1} / 2]$, this process reduces to the one presented [2].

:ore is relatively linear and $C(M)$ is small, the range of C must be small. Experiments conducted by he authors indicate that compactness and linearity are opposing traits.

Akushskii et al. introduced the method of descent and lift to determine the true value of a critical :ore. Both this method, and the redundant modulus core calculation (introduced by the authors in Section IV), assume that the core function can separate integers with cores that are different but equal modulo $C(M)$. Akushskii et al. tacitly assume that if a and b are two integers in $[0,M)$ with $C(b) = C(a) + C(M)$, then $a < M/2$ and $b > M/2$. *This condition, though sufficient* to solve the problem of critical cores, may be too restrictive when considering the selection of weights w_i, cf. Section V.

A weaker concept is introduced, called m-separability. This notion resolves core ambiguities and will be used in Section V in the selection of core functions.

Definition. Let $s \geq 1$ *be a real number*. A core function $C(\cdot)$ is said to be s-separable if

$$\left| a-b \right| < \frac{M}{s} \text{ implies } \left| C(a) - C(b) \right| < C(M).$$

If a core function is m-separable, assume its range $[0,M)$ is divided into m consecutive disjoint subintervals, say $I_1, \ldots, I_m$, *each of length* M/m. Within each of these, cores are unambiguously defined; i.e. if a and b are in I_j, and $C(a) \equiv C(b) \pmod{C(M)}$, *then* $C(a) = C(b)$. Thus for $a \in I_j$, the mapping $|C(a)|_{C(M)} \rightarrow C(a)$ is well defined and can be implemented by table lookup.

The method given by Akushkii et al. consists of two stages: descent and lift. The descent stage iteratively discards a modulus and evaluates a core function on the remaining moduli set until a non-

If $0 \leq C(a) < C(M)$, (4) gives the actual value of $C(a)$; otherwise, the core has been reduced modulo $C(M)$.

Let

$$C_{min} = min\{C(a) : 0 \leq a < M\}$$

$$C_{max} = max\{C(a) : 0 \leq a < M\}$$

From Theorem 3, it follows that

$$C_{min} + C_{max} = C(M) - \Sigma\, \omega_j .$$

Furthermore, since $C(0) = 0$,

$$C_{min} \leq 0.$$

The range of values the core function takes in $[0,M)$ is $[C_{min}, C_{max}]$. If a core $|C(a)|_{C(M)}$ is calculated by (4) and $C_{max} - C(M) < |C(a)|_{C(M)} < C(M) + C_{min}$, then $C(a) = |C(a)|_{C(M)}$ and thus the core has been unambiguously determined. Otherwise, (4) yields an ambiguous result. Cores having $|C(a)|_{C(M)}$ in the intervals $[0, C_{max} - C(M)]$ or $[C_{min} + C(M), C(M)]$ are called *critical* cores. (It should be emphasized that the result of the evaluation of (4) is therefore self validating.) Moreover, most practical core functions will have some critical cores.

In selecting the weights for a core function, the trade between compactness of range and linearity can now be appreciated. Since (4) involves only modular arithmetic, it is best evaluated using hardware similar to that for other residue calculations. Thus $C(M)$ should be on the same order as the moduli of the system. On the other hand, the core should be relatively linear so that C_{min} is near 0 and C_{max} is near $C(M)$, minimizing the number of integers in $[0, M)$ having critical cores. But if the

Formula (3) follows by applying the definition of the core function (1) to B_i, and using the facts that $[B_i/m_j] = (B_i/m_j)$ for $j \neq i$, and $[B_i/m_i] = ((B_i - 1)/m_i)$.

Theorem 12 (*Chinese Remainder Theorem for Core Functions,* Akushskii et al.). If $a = (a_i)$ is an integer in *[0,M)*, then its core is given by

$$C(a) = \Sigma a_i C(B_i) - C(M) \cdot R(a).$$

Proof. Using the rank function as given by (2), and the orthogonal core basis given in (3), the core of a is evaluated as

$$C(a) = \sum_{j=1}^{n} w_j \left[\frac{a}{m_j}\right] = \sum_{j=1}^{n} w_j \left(\frac{a - a_j}{m_j}\right)$$

$$= \sum_{j=1}^{n} \frac{w_j}{m_j} \left(\sum_{i=1}^{n} a_i B_i - M \cdot R(a) - a_j\right)$$

$$= \sum_{i=1}^{n} a_i \left(B_i \sum_{j=1}^{n} \frac{w_j}{m_j} - \frac{w_i}{m_i}\right) - R(a) \sum_{j=1}^{n} w_j \hat{m}_j$$

$$= \sum_{i=1}^{n} a_i C(B_i) - R(a) C(M).$$

In a residue implementation the core can be evaluated modulo $C(M)$ as

$$\left| C(a) \right|_{C(M)} = \left| \Sigma a_i C(B_i) \right|_{C(M)} \qquad (4)$$

The next lemma gives an explicit representation of the elements of the orthogonal basis.

Lemma. The elements of the orthogonal basis are given by

$$B_i = \hat{m}_i \cdot \left| \frac{1}{\hat{m}_i} \right|_{m_i}, \quad i=1,\ldots,n.$$

Proof. Since $B_i \varepsilon [0,M)$ is divisible by m_j for all $j \neq i$, it can be written as $B_i = k\hat{m}_i$, where $0 \leq k < m_i$. Furthermore, $B_i - 1$ is divisible by m_i, and so $k\hat{m}_i \equiv 1 \pmod{m_i}$, or equivalently $k = |1/\hat{m}_i|_{m_i}$.

Substituting these values for the B_i into Theorem 11 gives

$$a = \left| \Sigma a_i \hat{m}_i \left| \frac{1}{\hat{m}_i} \right|_{m_i} \right|_M,$$

a slightly different formulation of the Chinese remainder theorem than given by Szabo' and Tanaka [4]. Note also that the rank function $R(a)$ is analogous to the function $A(a)$ defined by [4]

$$a = \Sigma \hat{m}_i \left| \frac{a_i}{\hat{m}_i} \right|_{m_i} - A(a) \cdot M$$

In general $A(a) \leq R(a)$.

The values the core function assumes at the orthogonal basis elements perform the role of a basis. This is given by the *Chinese Remainder Theorem for Core Functions* which expresses the core of an integer in the range of the RNS as an inner product of its residues with the "orthogonal core basis" $\{C(B_i)\}$. These coefficients are constants of the RNS and are precomputed as

$$C(B_i) = B_i \frac{C(M)}{M} - \frac{w_i}{m_i}. \tag{3}$$

III COMPUTATION OF THE CORE FUNCTION

The definition of the core function as given in Section I, though useful for the derivation of its properties, is not efficient for computations within the context of residue arithmetic. Its use requires decoding of the residue representation, n integer divisions, and finally an inner product of length n.

In this section it will be shown how the core function can be evaluated directly from the residue representation of an integer in the range of the RNS. This result, referred to as the *Chinese Remainder Theorem for Core Functions*, gives core values modulo the core of M. This leads to the presence of some ambiguous cases, called critical cores. Several methods are presented that permit the unambiguous evaluation of these cores.

Let $B_i = (\beta_j^i) \in [0,M)$ be defined by

$$\beta_j^i = \begin{cases} 1 & i=j \\ 0 & otherwise \end{cases}$$

The set $B_1, B_2, \ldots, B_n$ is called an orthogonal basis for the residue system. The nomenclature is justified in view of the form of the Chinese Remainder Theorem as stated below without proof.

Theorem 11 (Chinese Remainder Theorem). If $a = (a_i)$, then $a = |\Sigma\, a_i B_i|_M$.

The difference between $\Sigma\, a_i\, B_i$ and a is a non-negative integer multiple of M. This number $R(a)$ is called the rank of a and satisfies the relation

$$\Sigma\, a_i B_i - a = M \cdot R(a). \tag{2}$$

$$\tau_j = \frac{-m_j C(\frac{a}{m_j}) + C(a) - \sum_{i \neq j} w_i \left(\frac{m_j \tau_i - a_i}{m_i} \right)}{w_j}$$

Since $0 \leq \tau_j < m_j$, the right side may be reduced modulo m_j, giving the result.

Theorem 10 (Akush[illegible] et al.). Let $a \cong (a_i)\,(mod\, M)$ and let p satisfy $(C(M), p) = 1$. Then

$$|a|_p = \left| \sum_{i=1}^{n} \left| \frac{\dot{w}_i \hat{m}_i}{C(M)} \right|_p \cdot a_i + \left| \frac{M}{C(M)} \right|_p \cdot C(a) \right|_p$$

Proof. The result follows immediately from Theorem 1 upon reduction modulo p.

Theorem 10 provides a new approach to general scaling. If $a = (a_i)$ is to be scaled by p, then a/p is computed as

$$\left| \frac{a}{p} \right|_{m_i} = \left| \frac{a - |a|_p}{p} \right|_{m_i} \qquad i = 1, \ldots, n$$

each involve modular calculations consisting of a core function evaluation and an inner product of (a_i) with a fixed weight vector.

We begin with the following lemma.

Lemma. If $a = (a_i)$ is divisible by p and $a/p = (\tau_i)$, then

$$C(a/p) = \frac{C(a) - \Sigma\, w_i \left(\frac{p\tau_i - a_i}{m_i} \right)}{p}$$

Proof. Since $a = a/p + \cdots + a/p$ (p summands), by iteration of Theorem 2,

$$C(a) = p \cdot C(\frac{a}{p}) + \Sigma w_i \left(\frac{p\,\tau_i - a_i}{m_i} \right).$$

The result follows by solving for $C(a/p)$.

Theorem 9 (Akushskii et al.). Let $a = (a_i)$ be divisible by m_j (i.e., $a_j = 0$) and let $(w_j, m_j) = 1$. Then

$$\left| \frac{a}{m_j} \right|_{m_j} = \left| \, \left| \frac{1}{w_j} \right|_{m_j} \left(C(a) + \sum_{i \neq j} \left| \frac{w_i}{m_i} \right|_{m_j} \cdot a_i \right) \right|_{m_j}$$

Proof. Since $a_j = 0$, applying the lemma with $p = m_j$ gives

$$C(\frac{a}{m_j}) = \frac{C(a) - \sum_{i \neq j} w_i \left(\frac{m_j\, \tau_i - a_i}{m_i} \right) - w_j \tau_j}{m_j}$$

where (τ_i) is the RNS encoding of a/m_j. Thus

Since $C(M) \neq 0$, $C(c) \neq C(a) + C(b) + \Sigma w_i \varepsilon_i$.

The following theorem gives a test for sign detection. The theorem does not directly appear in the works of Akushskii et al., but is a useful and obvious consequence of Theorem 6. As usual in a signed RNS, the interval *(0,M/2)* represents positive numbers, and the interval *(M/2,M)* represents negative numbers.

Theorem 8. Let the moduli m_i and the core $C(M)$ be odd, and let (a_i) be the residue representation of a non-zero integer $a \in (0,M)$. Then a represents a positive integer if and only if

$$(|2\alpha_1|_{m_1}, \ldots, |2\alpha_n|_{m_n})$$

is even.

Corollary. Let the moduli m_i and the core $C(M)$ be odd, and let $a \neq b$ be integers of the same sign in $(0,M)$ with residue representations (a_i) and (β_i). Then $b > a$ if and only if

$$(\,|2(\beta_1 - \alpha_1)|_{m_1}, \ldots, |2(\beta_n - \alpha_n)|_{m_n})$$

is even.

The next two theorems provide formulas for scaling a residue number by a modulus, and extension of base. Scaling by a modulus (or a product of several moduli) has been considerably simpler than general scaling, and the primary computational complexity has been the extension of base relative to the modulus (moduli) used for scaling, fundamentally a mixed radix conversion process. A mixed radix conversion relative to n moduli is performed in n-1 stages and no general techniques are known for collapsing the process into fewer stages. The algorithms given in the following theorems

Theorem 6 (Akushskii et al.). Let the moduli m_i be odd and let (a_i) and (β_i) be the residue representations of integers $a, b \in [0,M)$. Then $a + b$ overflows the system if and only if

i) $(|a_i + \beta_i|_{m_i})$ is odd and a and b have the same parity; or

ii) $(|a_i + \beta_i|_{m_i})$ is even and a and b have differing parity.

The second method for overflow detection under addition requires the computation of the core of the "sum" in two ways: overflow occurs when the results differ. This theorem does not require that the moduli be odd.

Theorem 7 (Akushkii et al.). Suppose $C(M) \neq 0$ and let $a, b, c \in [0,M)$ be given by $a = (a_i)$, $b = (\beta_i)$, $c = (|a_i + \beta_i|_{m_i})$. Then the sum $a + b < M$ if and only if

$$C(c) = C(a) + C(b) + \Sigma\, w_i \varepsilon_i ,$$

where

$$\varepsilon_i = \left[\frac{\alpha_i + \beta_i}{m_i} \right] = 0 \; or \, 1.$$

Proof. Clearly $a + b < M$ if and only if $a + b = c$, so the forward direction follows immediately from Theorem 2. For the converse, assume $a + b \geq M$, so that $c = a + b - M$. Then by Theorem 2 and its corollaries

$$C(c) = C(a+b-M) = C(a+b) - C(M)$$

$$= C(a) + C(b) + \Sigma\, w_i \varepsilon_i - C(M).$$

$$\sum_{i=2}^{n} w_i \delta_i$$

have the same parity.

Proof. Let

$$\varepsilon_i = \left[\frac{2\beta_i}{m_i} \right]$$

so that $\varepsilon_1 = \beta_1$. By Theorem 2,

$$C(a) = C(\frac{a}{2} + \frac{a}{2}) = 2\,C(\frac{a}{2}) + \Sigma\, w_i \varepsilon_i$$

so that

$$C(a/2) = \frac{C(a) - \sum_{i=2}^{n} w_i \varepsilon_i - w_1 \beta_1}{2}$$

From the definition of ε_i, $a_i = 2\beta_i - \varepsilon_i m_i$. Since for $i > 1$, m_i is odd, $\varepsilon_i = 0$ if a_i is even and $\varepsilon_i = 1$ if a_i is odd. Thus $\varepsilon_i = \delta_i$ is the parity function of a_i for $i = 2, \ldots, n$. Thus if $\beta_1 = 0$, $C(a)$ and

$$\sum_{i=2}^{n} w_i \delta_i$$

have the same parity, and if $\beta_1 = 1$, they have differing parity since w_1 is odd.

Two theorems are given next for the detection of overflow under the addition of two numbers. The first uses the parity information from Theorem 5 as follows. If all m_i (hence M) are odd and a sum $a + b$ is formed (where $0 \le a, b < M$), overflow occurs when $a + b \ge M$, and the resulting residue representation actually equals $a + b - M$. Hence overflow occurs exactly when the parity of the result modulo M is different from the parity expected, based on knowledge of the parity of a and b.

Proof. From Theorem 1, $C(M)\cdot a = M\cdot C(a) + \Sigma\, a_i w_i \hat{m}_i$. Since the moduli m_i and $C(M)$ are odd, a and $C(M)\cdot a$ have the same parity, $C(a)$ and $M\cdot C(a)$ have the same parity, and $\Sigma a_i\, w_i$ and $\Sigma\, a_i\, w_i\, \hat{m}_i$ have the same parity. Since a sum of two terms is even if and only if the summands have the same parity, the result follows.

A counterexample to the above theorem is next given in the case that the hypothesis that $C(M)$ be odd is dropped. Specifically, we show that if $C(M)$ is even, $C(a)$ and $\Sigma w_i\, \delta_i$ can have the same parity for odd a. This establishes the necessity of the hypothesis omitted in [1] and the error in the proof therein. Let $\{m_1, m_2\} = \{3,5\}$ and let $\{w_1, w_2\} = \{1,-1\}$. Then $C(15) = 2$ and $C(5) = 0$. The odd integer 5 is encoded as (2,0), and since both components are even, $\Sigma w_i\, \delta_i = 0$, completing the example.

Among other things, Theorem 4 shows that in the case where all the moduli are odd, scaling by 2 can be performed by checking the parity of $a = (a_i)$, subtracting 1 from each a_i if the parity is odd, and multiplying each a_i by the modulo m_i multiplicative inverse of 2. The computational requirements for checking parity thus consist of a core calculation, parity checks on the a_i, and the summation $\Sigma w_i\, \delta_i$.

In the case that one of the moduli is even, the parity of a is known immediately. Scaling by two for the odd moduli components is trivial, but calculation of the scaled result for the even modulus is normally performed by an extension of base. The next theorem uses the core to compute the residue of $a/2$ relative to the even modulus. Without loss of generality, the even modulus is assumed to be 2. (This theorem is actually a special case of Theorem 9 below.)

Theorem 5. (Akushskii et al.). Let $m_1 = 2$, let $a = (a_i)$ be even (so $a_1 = 0$), and let w_1 be odd. Let $a/2 = (\beta_i)$ and let δ_i be the parity function on a_i. Then $\beta_1 = 0$ if and only if $C(a)$ and

Theorem 3 (Akushskii et al.). $C(M - a - 1) = (C(M) - \Sigma w_i) - C(a)$.

Proof. Write $a = x_i m_i + a_i$ for each $i = 1,\dots,n$, where $0 \le a_i < m_i$. Then

$$M - a - 1 = (\hat{m}_i - x_i - 1)\, m_i + (m_i - a_i - 1),$$

and

$$0 \le m_i - a_i - 1 < m_i \quad for\, i = 1,\dots,n.$$

Thus

$$C(M - a - 1) = \Sigma\, w_i \left[\frac{M - a - 1}{m_i}\right]$$

$$= \Sigma\, w_i (\hat{m}_i - x_i - 1)$$

$$= C(M) - C(a) - \Sigma\, w_i.$$

The next theorem characterizes the parity of a residue encoded number in the case that all the moduli are odd. This theorem provides the basis for sign detection, magnitude comparison, and detection of overflow under addition. It should be noted that both the hypotheses and the proof of the converse direction of the equivalence are incorrect in [1], where the statement of the theorem does not require that $C(M)$ be odd. (A counterexample in the case that $C(M)$ is even is given below.)

Theorem 4 (Akushskii et al.). Assume the moduli m_i are odd and the core coefficients w_i are chosen so that the core $C(M)$ is odd. Let $a \equiv (a_i)\ (mod\ M)$, and let δ_i be the parity function on a_i (i.e., $\delta_i = 0$ *if* a_i is even, $\delta_i = 1$ *if* a_i is odd). Then a is even if and only if $C(a)$ and $\Sigma w_i \delta_i$ have the same parity.

Proof.

$$C(a+b) = \Sigma w_i \left[\frac{a+b}{m_i} \right]$$

$$= \Sigma\, w_i \left(\frac{a+b-\alpha_i-\beta_i+\varepsilon_i m_i}{m_i} \right)$$

$$= \Sigma\, w_i \left(\frac{a-\alpha_i}{m_i} \right) + \Sigma\, w_i \left(\frac{b-\beta_i}{m_i} \right) + \Sigma\, w_i \varepsilon_i$$

$$= C(a) + C(b) + \Sigma\, w_i \varepsilon_i.$$

Corollary. $C(-a) = -C(a) + \Sigma\, w_i \varepsilon_i$, where

$$\varepsilon_i = \left[\frac{-\alpha_i}{m_i} \right] = -1 \text{ or } 0.$$

Corollary. $C(a-b) = C(a) - C(b) + \Sigma\, w_i\, \varepsilon_i$, where

$$\varepsilon_i = \left[\frac{\alpha_i - \beta_i}{m_i} \right] = -1 \text{ or } 0$$

Corollary. If $\alpha_i \cdot \beta_i = 0$ for $i = 1,\dots,n$, then $C(a+b) = C(a) + C(b)$. In particular, $C(\hat{m}_i + \hat{m}_j) = C(\hat{m}_i) + C(\hat{m}_j)$ for $1 \le i \ne j \le n$.

The next result characterizes the symmetry of the core and can be interpreted as saying that the core is approximately odd in $[0,M)$ with respect to the midpoint of the interval. This theorem will be used to derive a bound on the range of $C(\cdot)$ on $[0,M)$, important for the selection of the w_i.

The following examples are given to illustrate the techniques of Akushskii et al. The moduli set chosen for all examples is $m_1 = 7$, $m_2 = 9$, $m_3 = 11$, and thus $M = 693$. The coefficients of the core function are $w_1 = -1$, $w_2 = -1$, and $w_3 = 3$. The orthogonal basis elements are $B_1 = 99$, $B_2 = 154$, and $B_3 = 441$. The cores $C(M) = 13$, and $C(B_1) = 2$, $C(B_2) = 3$, and $C(B_3) = 8$ are computed from the definition (1). Using (4), the core of $a = (a_1, a_2, a_3)$ is given by

$$|C(a)|_{13} = |2a_1 + 3a_2 + 8a_3|_{13} \tag{6}$$

For $0 \leq a < 693$, the minimum core is $C_{min} = -2$, and the maximum core is $C_{max} = 14$. Hence the only critical cores as computed by (6) are $|C(a)|_{13} = 0, 1, 11$ or 12.

Example 1. Determine the parity of $a = (2,1,1)$.

From (6), $|C(a)|_{13} = 2$. Since this is a non critical core it follows that $C(a) = 2$. The sum $\Sigma w_i \delta_i = 2$, where δ_i is the parity of a_i. Since $\Sigma w_i \delta_i$ and $C(a)$ have the same parity it follows from Theorem 4 that is a is even. (The decoded value of a is 100.)

Example 2. Determine the sign of $a = (3,5,5)$.

Let $|2 \cdot a|_M = (|2 \cdot 3|_7, |2 \cdot 5|_9, |2 \cdot 5|_{11}) = (6,1,10)$. The core of $|2 \cdot a|_M$ is 4. The sum $\Sigma w_i \delta_i$, where δ_i is the parity of $|2a_i|_{m_i}$, is odd. Hence $|2 \cdot a|_M$ is odd and so it represents a negative integer. (The decoded value of a is $500 \in (M/2, M)$, representing the M-complemented integer $a - M = -193$.)

Example 3. Extend $a = (6,8,4)$ to base $p = 5$.

From (6), $C(a) = 3$. Then using Theorem 10

$$|a|_5 = \left| \left| \frac{(-1)\cdot\ 99}{13} \right|_5 \cdot\ 6 + \left| \frac{(-1)\cdot\ 77}{13} \right|_5 \cdot\ 8 + \left| \frac{3.63}{13} \right|_5 \cdot\ 4 + \left| \frac{693}{13} \right|_5 \cdot\ 3 \right|_5 = 0$$

(The decoded value of a is 125.)

Example 4. Compute the core of $a = (1,8,8)$.

From (6), $|C(a)|_{13} = 12$, a critical core. The method of descent and lifting is applied.

Decent. The core of (1,8) relative to $m_1 = 7$, $m_2 = 9$ is computed for the core coefficients $w_1 = w_2 = 1$. The orthogonal basis elements are computed as $B_1 = 36$, $B_2 = 28$ with cores $C(B_1) = 9$, $C(B_2) = 7$, and $C(M) = 16$. Then

$$C(1,8) = |9\cdot\ 1 + 7\cdot\ 8|_{16} = 1.$$

This is not a critical core since the 2-moduli system has no critical cores.

Lifting. Theorem 10 is applied with $p = 11$, observing that $|1/C(M)|_{11} = |1/16|_{11} = 9$. Then β, the extension of (1,8) to the base $p = 11$ is given by

$$\beta = |1\cdot\ 9\cdot\ 9|_{11}\cdot\ 1 + |1\cdot\ 7\cdot\ 9|_{11}\cdot\ 8 + |63\cdot\ 9|_{11}\cdot\ 1 \ (mod\ 11)$$

$$= |4\cdot\ 1 + 8\cdot\ 8 + 6\cdot\ 1|_{11} = 8.$$

Equation (5) is then solved to obtain $k = 0$. Since C is 11-separable, this implies that $a = (1,8,8)$ is in the first subinterval of [0,693) and hence $C(a) = |C(a)|_{C(M)} - C(M) = 12 - 13 = -1$. (The decoded value of a is 8, the core of which can be directly computed from the definition (1), yielding the same result.)

IV. REDUNDANT MODULUS CORE CALCULATION

The use of the core function for performing the difficult RNS operations is practical only if the core function can be efficiently evaluated. While the method of Akushskii et al. is efficient for non-critical cores, the methods for the unambiguous evaluation of critical cores can be quite cumbersome. In particular, critical core evaluation could not be embedded in a flow-through architecture since the length of the computational path depends upon intermediate results. In this section we introduce a redundant modulus which eliminates critical cores altogether. Calculations modulo this redundant modulus must be performed for all computations upstream of a required core evaluation, so that when the core evaluation *C(a)* is required, the residue of a modulo the redundant modulus is known. The core evaluation is then simply an inner product of the residue components of a with fixed weights, where the arithmetic is performed modulo the redundant modulus.

Let C_{min} and C_{max} be the minimum and maximum cores on *[0, M)* as before, and choose a new, redundant modulus $m_{n+1} > C_{max} - C_{min}$ with $(m_{n+1}, m) = 1$ for $i = 1, \ldots, n$. Then for $a \in [0, M)$, $|C(a)|_{m_{n+1}}$ will uniquely determine $C(a)$ by

$$C(a) = \begin{cases} |C(a)|_{m_{n+1}} & \text{if } |C(a)|_{m_{n+1}} < m_{n+1} + C_{min}, \\ \\ |C(a)|_{m_{n+1}} - m_{n+1} & \text{otherwise.} \end{cases}$$

As shown in Theorem 1,

$$M \cdot C(a) = \Sigma\, a_i(-w_i)\hat{m}_i + a \cdot C(M).$$

Solving this equation for $C(a)$ in the ring of integers modulo m_{n+1}, it follows that

$$|C(a)|_{m_{n+1}} = \left| \Sigma\, a_i \cdot \left| \frac{-w_i}{m_i} \right|_{m_{n+1}} + a_{n+1} \cdot \left| \frac{C(M)}{M} \right|_{m_{n+1}} \right|_{m_{n+1}} \qquad (7)$$

All coefficients can be precomputed, and thus the core is calculated as a modulo m_{n+1} inner product of $(a_1, \ldots, a_n, a_{n+1})$ with a fixed vector.

Next note that the extended RNS on $\{m_1, \ldots, m_n, m_{n+1}\}$ uniquely represents integers in the interval $[0, M \cdot m_{n+1})$. We emphasize that (7) is guaranteed to uniquely determine $C(a)$ only if $a = (a_1, \ldots; a_n, a_{n+1}) \in [0, M)$. By chosing the redundant modulus to be larger than the length of the range interval of C on an expanded domain (e.g., $[0, 2M)$), all cores of integers in this new domain can be unambiguously computed as well.

With the provision that $a \in [0, M)$, redundant modulus core calculation can be used directly in the algorithms for decoding (Theorem 1), parity determination (Theorems 4 and 5), scaling by a modulus (Theorem 9), and extension of base (Theorem 10), and highly non-linear cores will suffice for these applications. However, Theorems 6 and 8 (overflow under addition and sign detection) require the capability to compute $C(|a|_M)$ for $a \notin [0, M)$, as they take advantage of parity changes upon wraparound modulo M. In the redundant RNS, wraparound will not occur under the addition of two integers $a, b \in [0, M)$, and the result will be uniquely represented as an integer in $[0, 2M) \subseteq [0, M \cdot m_{n+1})$. New algorithms are formulated below for applying the redundant modulus core calculation to these problems. A relatively linear core function is required for practical implementation.

Let C_{min} and C_{max} be as above and let C^*_{min} and C^*_{max} be the minimum and maximum values of C on $[M, 2M)$, respectively. (*Thus* $C^*_{min} = C_{min} + C(M)$ *and* $C^*_{max} = C_{max} + C(M)$.) The proof of the following theorem is trivial.

Theorem 13. Let $a, b \in [0, M)$. Then $a + b$:

i) overflows $[0, M)$ if $C(a+b) > C_{max}$, and

ii) does not overflow $[0, M)$ if $C(a+b) < C^*_{min}$.

If $C^*_{min} \leq C(a+b) \leq C_{max}$, $a+b$ may either overflow or underflow.

The ambiguous case is analogous to the ambiguity present in the methods of Akushskii et. al. when a critical core arises. However, far fewer ambiguous cases arise in the application of Theorem 13 than in the application of overflow detection using Theorem 6 or 7 above. For Theorems 6 and 7, $C(a)$, $C(b)$, and $C(|a+b|_M)$ must be evaluated. Thus the following cases are ambiguous, which can be resolved by the method of descent and lifting.

(i) $C(a) \leq C_{max} - C(M)$ or $C(a) \geq C(M) + C_{min}$ (i.e., the core of a is critical)

(ii) $C(b) \leq C_{max} - C(M)$ or $C(b) \geq C(M) + C_{min}$ (i.e., the core of b is critical)

(iii) $C(a+b) \leq C_{max} - C(M)$

(iv) $C(M) + C_{min} \leq C(a+b) \leq C_{max}$

(v) $C(a+b) \geq 2 \cdot C(M) + C_{min}$

(Conditions (iii) - (v) are equivalent to the statement that $C(|a+b|_M)$ is critical.) Of these possible five ambiguous cases, only (iv) remains an ambiguous case for Theorem 13. For this case, the method of descent and lifting can be applied as well.

Note next that Theorem 13 requires significantly fewer calculations than either Theorem 6 or Theorem 7, since only $C(a+b)$ must be computed and compared with two precalculated constants. Moreover, no restrictions on the parity of the m_i or $C(M)$ are required.

Finally, to apply Theorem 13 using the redundant modulus core calculation, m_{n+1} must be chosen satisfying $m_{n+1} \geq C^*_{max} - C_{min}$, so that all cores of integers in the interval $[0, 2M)$ can be computed.

We next develop algorithms for sign detection and magnitude comparison using the redundant core calculation. These algorithms also involve some ambiguous cases, but these cases are shown to coincide with those cases for which the problem is ill-conditioned. If the RNS represents signed integers, calculation of the sign of a is ill-conditioned if a is near $M/2$, and comparison of two numbers a and b is ill-conditioned if either a or b is near $M/2$. The dynamic range of an RNS should be sufficiently large so that the computational outputs can be unambiguously interpreted, and a buffer zone around $M/2$ should be maintained. Calculations should avoid this interval, since small perturbations of numbers in this interval can cause large changes in the values they represent.

Theorem 14. For $a \in [0, M)$, if $C(2a) < C^*_{min}$ then $a \in [0, M/2)$, and if $C(2a) > C_{max}$, then $a \in (M/2, M)$. If $C^*_{min} \leq C(2a) \leq C_{max}$, no conclusion can be drawn.

Proof. If $C(2a) < C^*_{min}$ then $2a \in [0, M)$. If $C(2a) > C_{max}$, then $2a \in [M, 2M)$.

The buffer zone around $M/2$ which should be avoided can now be given explicitly as

$$C^{-1}\{a \mid C^*_{min} \leq C(2a) \leq C_{max}\}.$$

For example, for the core function shown in Figure 1(a) ($M = 693$), this buffer zone is the interval [264, 434]. As before, use of the redundant core calculation for Theorem 14 requires

$m_{n+1} \geq C^*_{max} - C_{min}$.

To perform comparison of two numbers, the following lemma is required.

Lemma. For any integers a and q, $C(qM + a) \equiv C(a) \pmod q$.

Proof. By Theorem 2, $C(qM + a) = C(qM) + C(a) = q\,C(M) + C(a) \equiv C(a) \pmod q$.

If $a = (a_1, \ldots, a_{n+1})$, $b = (\beta_1, \ldots, \beta_{n+1}) \in [0, M)$ and $b - a < 0$, then in the redundant RNS $(\beta_i - a_i) = m_{n+1} \cdot M + (b-a)$. Hence by the lemma, calculation of the core of $(\beta_i - a_i)$ modulo m_{n+1} will uniquely determine $C(b-a)$.

Algorithm for Comparison. Let $a, b \in [0, M)$, with $[0, M/2)$ representing positive numbers and $(M/2, M)$ representing negative numbers, as usual. Compute the signs of a and b using Theorem 14. If the signs are different, the comparison is complete. Otherwise, compute the sign of $b - a$ to complete the comparison.

Notice that $b - a$ will be near $M/2$ only if

(i) one of a or b is near $M/2$ and the other is near 0 or M, or

(ii) neither is near $M/2$ and one is positive and the other negative.

Comparison of a and b fails only under condition (i), an ill-conditioned case.

Example 5. Compute the core of $a = (a_1, a_2, a_3, a_4) = (1, 8, 8, 8)$ using the same non-redundant moduli and core function for the examples of Section III, and using a redundant modulus of $m_4 = 32$.

Using formula (7), the core of $a \in [0,M)$ is given by

$$|C(a)|_{32} = |23 \cdot a_1 + 25 \cdot a_2 + 23 \cdot a_3 + 25 \cdot a_4|_{32}.$$

Hence $C(a) = |607|_{32} = 31$, which uniquely determines $C(a) = |C(a)|_{32} - 32 = -1$. Thus a critical core is avoided and the descent and lift algorithm is unnecesary.

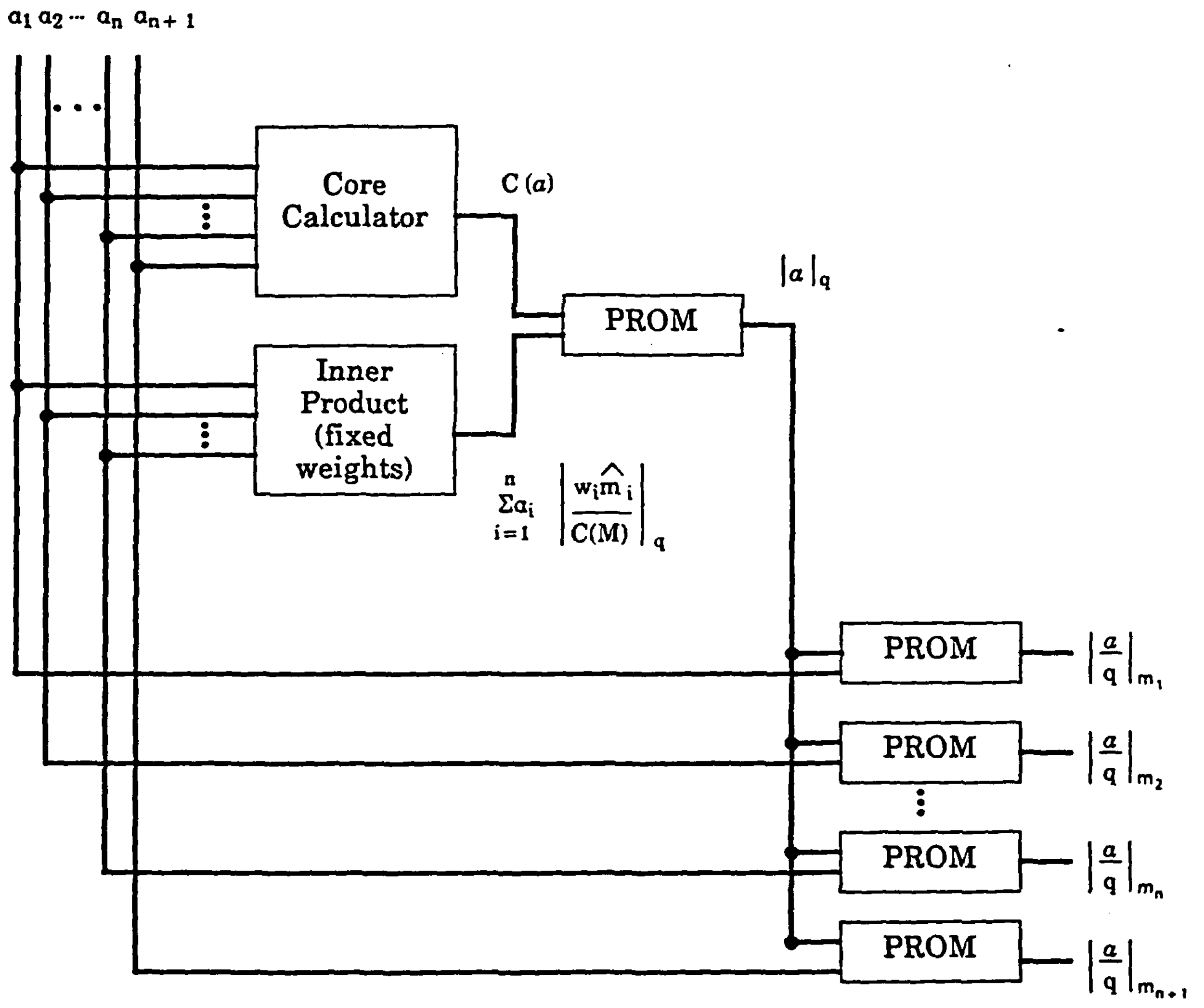
Functional hardware designs for performing scaling and thresholding using the redundant modulus core calculator are next discussed. These are included to give an appreciation of the hardware simplicity and short latency times for performing the "difficult" RNS operations using cores. Similar designs for other operations can be readily conceived.

These designs make use of several custom VLSI circuits designed and currently being fabricated by the authors, to be reported elsewhere[1]. These circuits permit calculations of the form $\Sigma\, c_i a_i \pmod m$ for fixed weights c_i and as many as six inputs a_i. The calculation is performed in a single 50 nsec clock cycle. Thus these circuits permit a one-cycle calculation of the core and a one-cycle calculation of the other inner products required for the theorems above.

[1] Design services and silicon compiler provided by Seattle Silicon Technology Incorporated

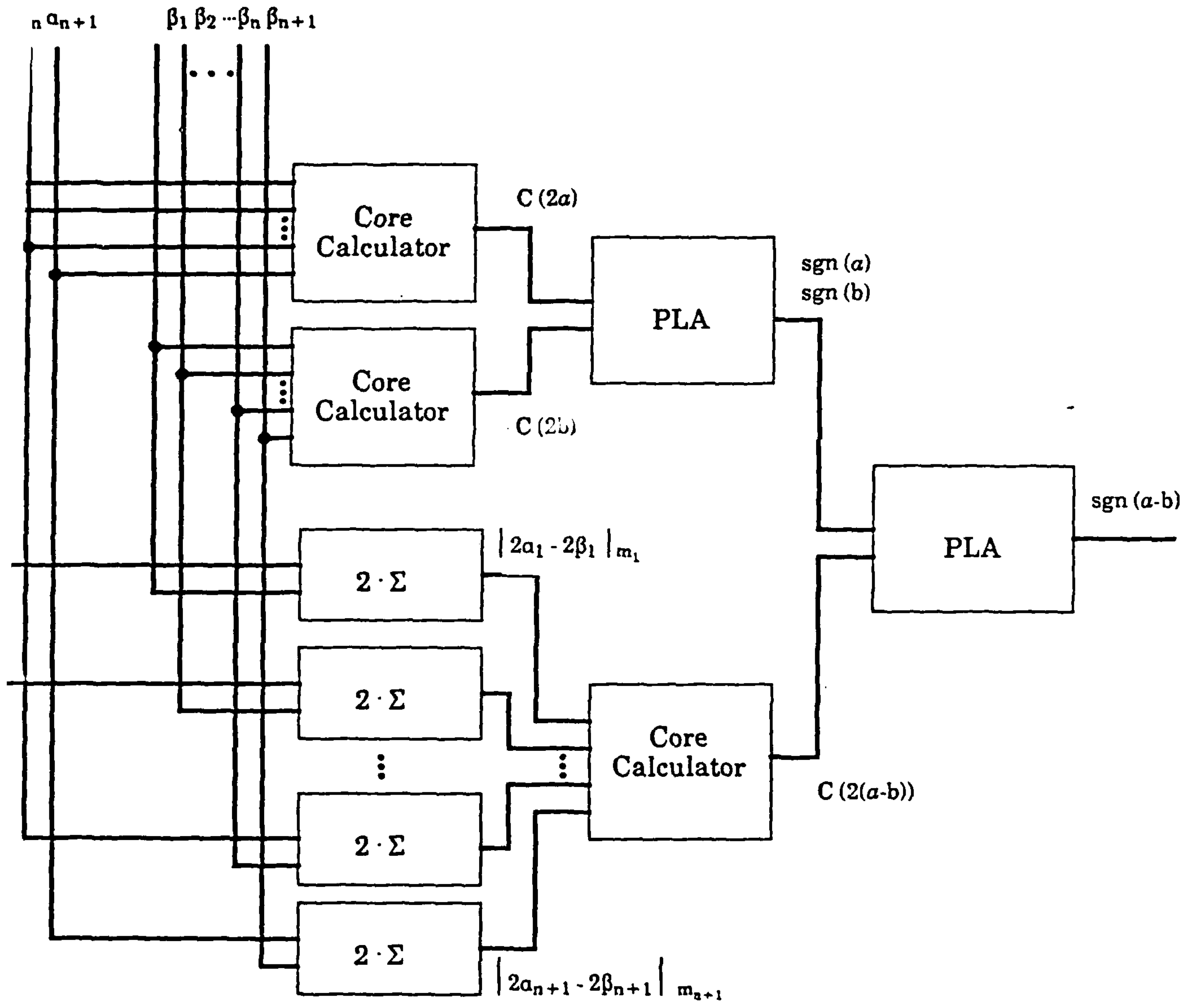
Shown in Figure 2 is a hardware architecture for scaling by a fixed integer q with $(q, m_i) = 1$. Scaling is performed by extension of base to the modulus q, subtraction (yielding an integer evenly divisible by q), and multiplication by the multiplicative inverse of q. The residue components (α_i) are simultaneously input to a core calculator and an inner product calculator which computes the modulo q inner product required in Theorem 10. The outputs of these two devices are then input to *a programmable read-only memory* (PROM) which completes the extension of base. This result is then fed to a PROM which computes a subtraction followed by a multiplication with the multiplicative inverse of q. Thus $n+4$ devices are required, and the latency of the calculation is three clock cycles. The procedure can be pipelined to produce a new scaled result each clock perod.

Figure 3 illustrates an architecture for comparison of two integers $a = (\alpha_i)$, $b = (\beta_i) \varepsilon [0,M)$. In the first clock period, the cores $C(2a)$ and $C(2b)$ and the residue representation of $2(a-b)$ are calculated. In the second clock cycle, the algebraic signs $sgn(a)$ and $sgn(b)$ are found as in Theorem 14 using a *programmable array logic* (PAL) device, and the core $C(2(a-b))$ is formed. In the final clock cycle the logic of the algorithm for comparison is implemented in a PAL.



$a_1\ a_2 \cdots a_n\ a_{n+1}$
Core Calculator
$C(\alpha)$
PROM
$|\alpha|_q$
Inner Product (fixed weights)
$\sum_{i=1}^{n} a_i \left| \frac{w_i \hat{m}_i}{C(M)} \right|_q$
PROM
$\left|\frac{\alpha}{q}\right|_{m_1}$
PROM
$\left|\frac{\alpha}{q}\right|_{m_2}$
PROM
$\left|\frac{\alpha}{q}\right|_{m_n}$
PROM
$\left|\frac{\alpha}{q}\right|_{m_{n+1}}$

Figure 2. Scaling an integer $\alpha = (a_1, ..., a_{n+1}) \in [0, M)$ by a fixed scale factor q requires $n + 4$ integrated circuits, has a latency of 3 clock periods, and can be pipelined with throughputs of 20 MHz.



igure 3. Comparison of integers $a = (a_1, \ldots, a_{n+1})$, $b = (\beta_1, \ldots, \beta_{n+1}) \in [0,M)$ requires n + 5 integrated circuits, and has a 3 clock period latency and 20 MHz throughput.

V. SELECTING A CORE FUNCTION

Practical implementation of residue class core functions is contingent upon creating a methodology that finds an appropriate function. For a given moduli set, such a method should lead to a core function which solves problems such as parity and sign determination, allows scaling and basis extension, but which has small enough range to make its evaluation practical.

Akushskii et al. discuss some partial solutions to the selection of the core. Nevertheless, the general problem is not undertaken. In this section a general methodology is presented which allows one to find the core which is "optimal" for the application at hand. First a decomposition of the core function is introduced and used to find an upper bound of the core range. A summary is then given of all conditions the core function should satisfy to maximize its utility. Finally the selection of core functions is formulated as an integer optimization problem.

The analysis of the variability of a core function is possible by first decomposing it into a linear part and a periodic part of period M. Let

$$C(a) = L(a) + P(a)$$

where

$$L(a) = \Sigma\, w_i (a/m_i) = a\, C(M) / M$$

$$P(a) = -\Sigma\, (w_i a_i / m_i)$$

the weights of the core function are chosen so that *C(M)* is positive, the linear part is increasing and s minimum and maximum are attained at 0 and *M-1* respectively. The extrema of the periodic part ollow immediately by separating the weights w into groups according to their sign.

$$P_{min} = \sum_{i \varepsilon I_+} (-w_i) \frac{m_i - 1}{m_i},$$

$$P_{max} = \sum_{i \varepsilon I_-} (-w_i) \frac{m_i - 1}{m_i},$$

vhere $I_+ = \{1 \leq i \leq n : w_i > 0\},$

$I_- = \{1 \leq i \leq n : w_i < 0\}.$

Bounds for the extrema and range of the core function follow immediately from the decomposition. These results shall be used to define the optimization criterion to find an applicable set of weights.

Theorem 15. The extrema and range of the core function satisfy the following inequalities

$$C_{min} \geq -[-P_{min}],$$

$$C_{max} \leq \left[P_{max} + \frac{M-1}{M} \cdot C(M) \right],$$

$$range\, of\, C \leq \left[P_{max} + \frac{M-1}{M} C(M) \right] + \left[-P_{min} \right].$$

Lemma: Let $0 \leq a < b < M$. If $C(b) - C(a) \geq C(M)$, then

$$b - a \geq \frac{M}{C(M)} (C(M) - P_{max} + P_{min}).$$

of. For any $a \in [0, M]$,

$$\frac{a}{M} C(M) + P_{min} \leq C(a) \leq \frac{a}{M} C(M) + P_{max},$$

quivalently

$$(C(a) - P_{max}) \frac{M}{C(M)} \leq a \leq (C(a) - P_{min}) \frac{M}{C(M)}.$$

lemma then follows by using the upper bound for a and the lower bound for b.

eorem 16. The core function C is m-separable if

$$\frac{C(M) - P_{max} + P_{min}}{C(M)} > \frac{1}{m}$$

e quotient shown in the left hand side of Theorem 16 may be considered as an index which ovides a measurement of linearity of the core function. This quotient varies between $-\infty$ and 1. It ces values close to 1 when $P_{max} - P_{min}$ is close to 0, i.e., $C(\cdot)$ is close to its linear part $L(\cdot)$.

e last two theorems and the results in Section II are now used to define the criterion and nditions for the selection of a core function. In Section IV the redundant modulus is chosen larger an the range of the core function. It follows that a practical core must have a small range, so that optimal core function is defined as one having minimal range over all core functions for a fixed oduli set. The last inequality in Theorem 15 is used to define the functional to be minimized. The thors' experience indicates that the bound for the range is tight when close to the optimal ution and that the process actually does lead to core a with minimal range. The conditions the re function should satisfy depend on the particular application at hand. The ones presented

Step (i) Verify $b \neq 0$.

Step (ii) Compare r_i and b. If $r_i < b$, $[a/b] = q_i$ and the algorithm is complete.

Step (iii) Write

$$\frac{r_i}{b} = \frac{S_L(r_i)\cdot r_i}{S_L(r_i)\cdot b}$$

$$= S_{L_1}(S_L(r_i)\cdot b)\cdot \frac{S_L(r_i)\cdot r_i}{S_{L_1}(S_L(r_i)\cdot b)\cdot S_L(r_i)\cdot b}.$$

Define $q_{i+1} = q_i + S_{L_1}(S_L(r_i)\cdot b)$, and $r_{i+1} = a - q_{i+1}\cdot b$. Return to step (i).

rove convergence of the algorithm, we need only show that $1 \leq S_{L_1}(S_L(r_i)\cdot b) \leq [r_i / b]$. By ıition, $S_{L_1} \geq 1$ always. If $S_L(r_i)\cdot b > L_1$, then $S_{L_1}(S_L(r_i)\cdot b) = 1$. Otherwise

$$S_{L_1}(S_L(r_i)\cdot b)\cdot S_L(r_i)\cdot b \leq L_1 < S_L(r_i)\cdot r_i$$

ce the quotient

$$\frac{S_L(r_i)\cdot r_i}{S_{L_1}(S_L(r_i)\cdot b)\cdot S_L(r_i)\cdot b} > 1$$

establishing convergence.

odification of the above algorithm is next presented which has faster convergence for relatively ır core functions. However, this algorithm is more difficult to analyze, and a general proof of ıergence has not been obtained in terms of the parameters involved. In the first division rithm, the sequence of quotient estimates q_i converged monotonically to $[a/b]$; in the next

$$s_L(c) = max\left\{\left\lceil \frac{L \cdot C(M)}{(c-P_{min}) \cdot M} \right\rceil, 1\right\}$$

ı Section VI, for any integer a, $0 \le a \le L$, $s_L(C(a)) \cdot a \le L$. We next define a function $S_L\,[0,L] \to N$ ⁄ iteratively in terms of s_L. Let $a_0 = a \in [0,L]$ be given. Let $c_i = C(a_i)$, $s_i = s_L(c_i)$, and define ' $= s_i a_i$. This procedure terminates when $s_{k+1} = 1$, $s_k > 1$, and we take

ı) $= s_1 s_2 \cdots s_k$, again giving $S_L(a) \cdot a \le L$. We claim also that $S_L(a) \cdot a > L/2 - \Delta P \cdot M / C(M)$. To prove , assume $s_L(c) = 1$. Then

$$\left\lceil \frac{L \cdot C(M)}{(c-P_{min}) \cdot M} \right\rceil \le 1,$$

$$2(c - P_{min}) \cdot M > L \cdot C(M)$$

l thus

$$c > \frac{L \cdot C(M)}{2M} + P_{min}.$$

ıs if x has $s_L(C(x)) = 1$,

$$x \ge (C(x) - P_{max}) M / C(M)$$

$$> \frac{L}{2} - \frac{\Delta P \cdot M}{C(M)}$$

ce by the definition of S_L, $s_L(S_L(a) \cdot a) = 1$, the claim is established.

ıume the redundant modulus is chosen such that all cores of integers in $[0,L]$ can be ambiguously computed by the redundant modulus method. Define $L_1 = L/2 - \Delta P \cdot M / C(M)$

ıorithm for D ision I. Let $a,b \in [0,M)$. The quotient $\lfloor a/b \rfloor$ is computed iteratively by the lowing steps. For initial conditions, take $q_0 = 0$, $r_0 = a$.

I. ALGORITHMS FOR DIVISION

any number system, division is the most computationally complex of the fundamental arithmetic ɔerations. In the RNS, division algorithms have been so complex that the system has been judged ısuitable for computational processes requiring division. RNS division algorithms have been ·esented by Szabo and Tanaka [4], and more recently, division algorithms using the core function ave been presented by Akushskii et al. [7]. However, even the core-based division algorithms are ttremely cumbersome, and cannot be utilized in a high-speed real time signal processing system.

ı this section, a new core-based algorithm for general division is presented. By previous standards, ıe algorithm is very efficient, and is competitive with the usual binary system algorithms for ivision. Two algorithms are presented: the first is guaranteed to converge for any RNS and core ınction, while the second has superior convergence properties for well behaved core functions. oth algorithms are iterative, and thus are perhaps not suited to flow-through architectures for high ɔeed signal processing. However, the algorithms might well be suited to a microcoded general urpose RNS arithmetic unit.

a core function is approximately linear, then for b bounded away from zero, $a/b \approx C(a) / C(b)$. loreover, since $P(a) = C(a) - (C(M) / M) \cdot a$ does not increase with a,

$$\lim_{k \to \infty} \frac{C(ka)}{C(kb)} = \frac{a}{b}.$$

/e will temporarily utilize the expanded dynamic range provided by the redundant modulus to nprove the estimate of a/b. At each stage of the algorithms, an improved estimate q of the uotient is found. The process terminates when $0 \leq a - qb < b$. As in the algorithms for omparison, core-derived upward scaling of an integer is required. We define a scaling function milar to those defined in Section VI.

et an integer $L > 0$ be given and let $C_L = max \{C(x) / 0 \leq x \leq L \}$. Define an non-increasing ınction $s_L : [C_{min}, C_L] \to N - \{0\}$ by

i	a_i	b_i	$C(a_i)$	$C(b_i)$	$t(c_i)$	a'_i	b'_i	$S(a'_i,b'_i)$
0	(3,8,10,11)	(5,1,1,13)	6	8	(2,1,6,2)	(1,7,4,9)	(3,0,6,11)	4
	395	397			226	169	171	
1	(4,1,5,4)	(5,0,2,12)	12	13	(0,6,7,2)	(4,4,9,2)	(5,3,6,10)	6
	676	684			546	130	138	
2	(3,6,10,12)	(2,0,3,28)	13	15	(4,5,5,23)	(6,1,5,21)	(5,4,9,5)	4
	780	828			599	181	229	
3	(3,4,9,20)	(6,7,3,20)	12	18				
	724	916						

Since $C(b_3) - C(a_3) = 6 > P_{max} - P_{min}$, $b > a$.

$$\geq \left(\frac{a\,C(M)}{M} + P_{min} - P_{max} \right) \frac{M}{C(M)}$$

$$a + (P_{min} - P_{max}) \frac{M}{C(M)}$$

Then

$$a' = a - t(C(a) \leq (P_{max} - P_{min}) \frac{M}{C(M)} ,$$

and

$$b' = b - t(C(a)) < 2\,(P_{max} - P_{min}) \frac{M}{C(M)} ,$$

since the condition on the cores implies that $/b\text{-}a\,/ < (P_{max} - P_{min})\; M/C(M)$. Thus a sufficient condition for $S(a', b') \geq 2$ is

$$4\,(\Delta P)\,\frac{M}{C(M)} \leq L$$

Example 7. Let the moduli set be {7,9,11} with core coefficients {-1, -1, 3}. Then $\Delta P = (6/7 + 8/9 + 30/11) \approx 4.47$. Since $5\ (\Delta P) > C(M) = 13$, the first algorithm cannot be used. For the second algorithm we must have

$$L \geq \frac{4\,(4.47) \cdot\ 693}{13} = 953,$$

so the redundant modulus $m_4 = 32$ will suffice. Since calculated cores of 30 and 31 represent true cores -2 and -1, respectively, L will be the largest number whose core does not exceed 29; viz, $L = 1517$.

We compare $a_0 = a = (3,8,10,11) = 395$ and $b = (5,1,1,13) = 397$.

Both functions t and s can be implemented by table lookup since their domain is small if the range of the core function is small.

We next iteratively define a function $S(a,b)$ for $a,b, \in [0,L]$ as follows. Take $a_0 = a$, $b_0 = b$. Let $c_i = min\ \{C(a_i), C(b_i)\}$ and $s_i = s(c_i)$, and define $a_{i+1} = s_i \cdot a_i$, $b_{i+1} = s_i \cdot b_i$. This procedure is iterated until $s_{k+1} = 1$, $s_k > 1$ *(usually only one or two iterations are needed).* Then $S(a,b) = s_0 s_1 \cdots s_k$. Thus $S(a,b) \cdot a \leq L$, $S(a,b) \cdot b \leq L$, and $min\ \{s(S(a,b) \cdot a), s(S(a,b) \cdot b)\} = 1$; i.e., based only on knowledge of the cores, $S(a,b) \cdot a$ and $S(a,b) \cdot b$ have been upward scaled as much as possible.

Second Algorithm for Comparison. Assume $C(M) > 0$ and let $a_0 = a = (a_i)$, $b_0 = b = (\beta_i) \in [0,L]$.

(i) If $a_i = \beta_i$ for i = 1,...,n, then $a = b$.

(ii) If $C(a_i) - C(b_i) \geq \Delta P$, then $a > b$ and if $C(b_i) - C(a_i) \geq \Delta P$ then $b > a$.

(iii) Let $c_i = min\ (C(a_i), C(b_i))$, and form $a_i' = a_i - t(c_i)$, $b_i' = b_i - t(c_i)$. (Then a_i' and b_i' are nonnegative.)

(iv) Compute $a_{i+1} = S(a_i', b_i') \cdot a_i'$, $b_{i+1} = S(a_i', b_i') \cdot b_i'$ and go to (ii).

A condition on L (and therefore also on m_{n+1}) is next derived which guarantees convergence of the algorithm. This is done by examining worst-case behavior. It can be readily seen that convergence is ensured if each $S(a_i, b_i) > 1$; then $|a_{i+1} - b_{i+1}| > |a_i - b_i|$, and the hypothesis of step (ii) will eventually be satisfied.

We assume, without loss of generality, that $C(a) < C(b)$ and $|C(a) - C(b)| < \Delta P$. Then

$$t(C(a)) \geq \left(C(a) - P_{max}\right) \frac{M}{C(M)}$$

algorithm is iterative and requires the use of a redundant modulus. The redundant modulus must be chosen sufficiently large to guarantee convergence.

As in Section IV the redundant modulus is chosen to satisfy $m_{n+1} > C_{max} - C_{min}$. The method of redundant modulus core calculation can then be used to unambiguously determine cores on an interval $[0,L] \supset [0,M)$, where L depends upon m_{n+1}. Let C_L be the maximum core on $[0,L]$. Given integers $a_i, b_i \in [0,L]$, the comparison algorithm will find integers $a_{i+1}, b_{i+1} \in [0,L]$ satisfying (i) $a_i \leq b_i$ if and only if $a_{i+1} \leq b_{i+1}$; and (ii) $|a_{i+1} - b_{i+1}| > |a_i - b_i|$. This is iterated until the first step k for which $|C(a_k) - C(b_k)| > \Delta P$. By the lemma above, the comparison can then be determined.

To obtain a_{i+1} and b_{i+1} from a_i and b_i, a subtraction and a scaling are performed. From the cores of a and b, a number is computed and subtracted from both a and b giving nonnegative results. From the cores of these differences, a scale factor greater than one is computed and multiplied by the differences. These results will be guaranteed to lie in $[0,L]$ and are taken as a_{i+1} and b_{i+1}. Hence we must define functions on cores to give the subtrahend and the scale factor. As mentioned above, it is required that $C(M) > 0$.

Let $\mathbf{Z}$ denote the integers and $\mathbf{N}$ the natural numbers. Define a non-decreasing function $t: [C_{min}, C_L] \to N$ by $t(c) = max\{[(c - P_{max}) \cdot M/C(M)], 0\}$. Then for any integer $a \geq 0$,

$$0 \leq t(C(a)) \leq t\left(\frac{C(M)}{M} \cdot a + P_{max}\right) = a.$$

Define a non-increasing function $s: [C_{min}, C_L] \to N - \{0\}$ by

$$s(c) = max\left\{\left[\frac{L \cdot C(M)}{(C - P_{min}) \cdot M}\right], 1\right\}.$$

Then for any integer a, $0 \leq a \leq L$,

$$s(C(a)) \cdot a \leq L \; since \; a \leq \frac{(C(a) - P_{min})M}{C(M)}.$$

If the cores are calculated by the redundant modulus method, then by the lemma preceding the algorithm of signed comparison in Section IV, $C(a_1 - \beta_1, \ldots, a_n - \beta_n, a_{n+1} - \beta_{n+1}) \equiv C(d) \pmod{m_{n+1}}$. Thus redundant modulus core calculation can be used for the algorithm.

To select a core function which allows this comparison algorithm, the linear condition $5 \cdot \Delta P < C(M)$ should be included in the list of linear conditions for the integer optimization problem formulated in Section V.

Example 6. Let the moduli set be {23,25,27,29,31} and the core weights be {-3,-2,4,-1,3} as in Figure 1(c). Then

$$\Delta P = \Sigma |w_i| \frac{m_i - 1}{m_i} \approx 12.51$$

and $C(M) = 67$. Since $5 \cdot \Delta P < 67$, the algorithm applies.

(i) *Compare* $a = (6,0,1,22,2)$ *and* $b = (7,0,5,23,10)$. Then $C(a) = 6$, $C(b) = 24$, $C(b) - C(a) = 18 > \Delta P$, so $b > a$. The values are $a = 1{,}000{,}000$, $b = 5{,}000{,}000$.

(ii) Compare $a = (6,0,1,22,2)$ and $b = (12,0,2,15,4)$.

Then $C(a) = 6$, $C(b) = 11$, so $|C(a) - C(b)| < \Delta P$. Form $d = (6,0,1,22,2) - (12,0,2,15,4) = (17,0,26,7,29)$. $C(d) = 58 \nleq 2 \cdot \Delta P + P_{max}$, so $b > a$. The values here are $a = 1{,}000{,}000$, $b = 2{,}000{,}000$.

Note that the condition $5 \cdot \Delta P < C(M)$ is a relatively strong linearity condition: while the core of Figure 1(c) satisfies this condition, the core of Figure 1(a) does not. The next algorithm for comparison relaxes this linearity condition at the cost of increased computational requirements. The

$$\leq \left(\frac{C(M)}{M} \cdot a + P_{max} \right) - \left(\frac{C(M)}{M} \cdot b + P_{min} \right)$$

$$= \frac{C(M)}{M} \cdot (a-b) + (P_{max} - P_{min}),$$

and hence $a\text{-}b \geq 0$. Since $C(a) \neq C(b)$, $a \neq b$, obtaining the strict inequality. To show (ii), first note that for any integer x,

$$\left(C(x) - P_{max} \right) \cdot \frac{M}{C(M)} \leq x \leq \left(C(x) - P_{min} \right) \frac{M}{C(M)} .$$

Thus

$$|a-b| \leq \left(|C(a) - C(b)| + (P_{max} - P_{min}) \right) \frac{M}{C(M)}$$

$$\leq \frac{2M}{C(M)} \left(P_{max} - P_{min} \right).$$

First Algorithm for Comparison. Assume $5 \cdot \Delta P < C(M)$. Then integers $a = (a_i)$, $b = (\beta_i) \in [0,M)$ can be compared via the following steps.

(i) Compute $C(a)$ and $C(b)$. If $C(a)\text{-}C(b) \geq \Delta P$, then $a > b$, or if $C(b) - C(a) \geq \Delta P$, then $b > a$. If either condition holds, the algorithm is complete.

(ii) Otherwise, $|C(a) - C(b)| < \Delta P$. Form the difference $d = (a_i - \beta_i) \in [0,M)$ and compute $C(d)$. If $C(d) \leq 2(P_{max} - P_{min}) + P_{max}$ then $b \leq a$; otherwise $b > a$.

The validity of step (i) of the algorithm follows immediately from the lemma. For step (ii), the lemma implies $|a\text{-}b| \leq 2M \cdot \Delta P / C(M)$. Thus d lies in one of the intervals $I_1 = [0, 2M \cdot \Delta P / C(M))$ or $I_2 = [M\text{-}2M \cdot \Delta P / C(M), M)$. Since $max \{(C(x) / x \in I_1\} \leq 2 \cdot \Delta P + P_{max}$ and $min \{C(x) / x \in I_2\} \geq C(M) - 2 \cdot \Delta P + P_{min}$, it follows from the algorithm hypothesis that $C(I_1) \cap C(I_2) = \emptyset$. Thus if $C(d) \in C(I_1)$, $d \in I_1$, and if $C(d) \in C(I_2)$, $d \in I_2$, establishing step (ii).

VI. ALGORITHMS FOR COMPARISON

In Section IV, an algorithm utilizing the *redundant modulus core* calculation was given for the comparison of two numbers in a signed RNS. The algorithm requires the capability to determine whether a number lies in the upper or lower half of $[0,M)$. This is accomplished by maintaining a buffer zone around $M/2$ so the determination can be made by an examination of the core. In this section, two new algorithms are presented for comparison in an unsigned RNS, and no restrictions on the numbers are assumed. The first algorithm is computationally simple: to compare a and b, only $C(a)$, $C(b)$, and $C(a-b)$ are required. However, a linearity condition on the core is required, viz., $5(P_{max} - P_{min}) < C(M)$. The second algorithm is iterative, and is guaranteed to terminate if $C(M) > 0$ and if the redundant modulus is chosen sufficiently large. Convergence can be hastened by increasing the degree of separability of the core or by increasing the redundant modulus.

We begin the development of the first algorithm with the following lemma. Recall that P_{min} and P_{max} were defined in Section V, and that

$$\frac{C(M)}{M} \cdot a + P_{min} \leq C(a) \leq \frac{C(M)}{M} \cdot a + P_{max} .$$

We define $\Delta P = P_{max} - P_{min}$, so

$$0 < \Delta P = \Sigma |w_i| \frac{m_{i-1}}{m_i} < \Sigma |w_i| .$$

Lemma. Let a and b be arbitrary integers, and let $C(M) > 0$.

(i) If $C(a) - C(b) \geq \Delta P$, then $a > b$

(ii) If $|C(a) - C(b)| < \Delta P$, then $|a-b| < 2M \cdot \Delta P / C(M)$.

Proof. For (i), observe that

$$P_{max} - P_{min} \leq C(a) - C(b)$$

Proof. (i) and (ii) follow immediately from the definition of S_n and from $C(0) = 0$.

(iii) Follows since $P_{min} + a.C(M)/M \leq -n$ for all $a \in S_n$.

(iv) Let $n > K = \lfloor -P_{min} \rfloor$, then $n \geq K + 1 > -P_{min}$, which in term imolies that S_n is empty.

Algorithm for search of C_{min}.

Step 1. Start algorithm with initial values $a_1 = c_1 = 0, n_1 = 1$, and $J_1 = S_1$.

Step 2. Given a_i with $C(a_i) = c_i = 1 - n_i$, evaluate the core of successive elements of the set $J_i = \{a_i + 1, a_i + 2, ..., M\} \cap S_{ni}$ until either (i) an integer b is found such that $C(b) = -K$, or (ii) an integer b is found such that $-K < C(b) < c_i$, or (iii) $C(x) \geq c_i$ for all $x \in J_i$. If (i) or (iii) occur then C_{min} has been found. $C_{min} = C(b)$ in case (i), and $C_{min} = C(a_i) = c_i$ in case (iii). In the event of case (ii), step 2 is repeated with $a_{i+1} = b, c_{i+1} = C(b)$, and $n_{i+1} = 1 - C(b)$.

Example. In the RNS with moduli set {7,9,11} consider the core function with weights {-1, -1,3}. Then $P_{min} = -30/11$, $K = 2$ and Ω is the set of multiples of 7 and 9. From *Theorem 17*, $\max S_1 < 92$, and $\max S_2 < 38$. The first iteration of step 2 evaluates the core of the elements in the set $J_1 = \{7,9,14,18,21,...,91\}$. Since $C(7) = -1$, then a second iteration of step 2 is started with $a_2 = 7$, $c_2 = -1$, $n_2 = 2$, and $J_2 = \{9,14,18,21,27,28,35,36\}$. Since $C(9) = -2 = -K$, it follows that $C_{min} = C(9) = -2$.

In the case of an RNS with small range the extrema of a core function can be found through an exhaustive search. For moderately large systems an exhaustive search may prove prohibitive. In the remainder of this section an algorithm is presented for the evaluation of the minimum of a core function. The authors' experience suggests that the method has fast convergence even for an RNS with large range. The algorithm is based on a search through a chain of dynamically chosen sets with successively smaller ranges. Since $C(0) = 0$, the algorithm starts by searching for an integer in the domain of the RNS, which has negative core. Since downward jumps of the core function occur only at multiples of the moduli m_i for which w_i is negative, i.e., $i \in I_-$, the search is restricted to this set. The first step is then to search through increasing multiples of those moduli until one with negative core is found. If this occurs, say $C(a_1) = -n_1$, then the algorithm goes into a second phase with a search through multiples larger than a_1 and with a core value less than $-n_1$. If such an integer is found, say a_2 with $C(a_2) = -n_2$, then the second phase is restarted replacing a_1 and n_1 with a_2 and n_2. The sets of integers for which the core is less than or equal to $-n$ forms a decreasing chain, so the successive phases of the algorithm take place in *successively smaller domains. This assures a fast* stopping rule for the algorithm and convergence to C_{min}.

Let $\Omega = \{ a \,/\, a_i = 0 \text{ for some } i \text{ a } I_- \}$, be the set of multiples of the moduli m_i for which w_i is negative. Let $S_n = \{a \in [0,M) \cap \Omega \,/\, C(a) \leq -n \}$, *for* $n \geq 0$, be the chain of sets through which the search takes place. The search algorithm is based on properties of the S-chain given in the theorem below.

Theorem 17. The chain of sets S_n satisfy the following properties:

(i) The minimum C_{min} is attained in S_0.

(ii) The chain is decreasing, i.e., $S_n \supset S_{n+1}$.

(iii) The maximum element of S_n is bounded above by $-M.\ (n + P_{min}) / C(M)$.

(iv) The chain terminates after K steps, where $K = [-P_{min}]$; i.e., $S_n = 0$, for all $n > K$.

These conditions for the optimization problem arise as follows. Condition (i) states that $C(M) > 0$. Condition (ii) is equivalent to the condition $C(M)$ odd, required for parity determination, Theorem 4. Condition (iii) states that w_i and m_i are relatively prime, necessary for scaling by m_i, Theorem 9. This condition is equivalent to the existence of a solution to the set of linear equations

$$w_i = x_{ij} p_{ij} + y_{ij}$$

where p_{ij} are the prime factors of m_i, and x_{ij} and y_{ij} are integers with $1 \leq y_{ij} < p_{ij}$. The restriction (iv) follows from the separability condition, Theorem 16, where K_1 is an appropriate constant, e.g., $K_1 = 1/m_1$ where m_1 the smallest modulus; will permit lifting and descent. Larger values for K_1 will permit greater separability. Finally the left hand of inequality (v) represents the largest jump the core function may take. This condition may be used to reflect prior restrictions on the range, e.g., if five bit representation is to be used then $K_2 = 32$.

The optimization problem can be solved by standard techniques in integer programming such as branch and bound and enumeration searches [5], [6]. Such methods were used for the case of the moduli set {23, 25, 27, 29, 31} obtaining a core function with weights {-3, -2, 4, -1, 3} (This function is m-separable for all $m \geq 2$).

Algorithm for evaluating core function extrema. In Theorem 15 bounds are given for C_{max} and C_{min}, which prove sufficiently tight for closely linear core functions. In many cases however more accuracy is desired and so exact values of the extrema of the core function are calculated.

From Theorem 3 it follows that if C_{min} is attained at a, then C_{max} is attained at $(M - a - 1)$, and that $C_{max} + C_{min} = C(M) - \Sigma w_i$. Hence it is sufficient to evaluate only one extreme value of the core function.

include all those required for the algorithms for the difficult RNS operations. Since the range bound given in Theorem 15 assumes that C (M) is positive, this condition is required. Parity determination, scaling and *m*-separability form the remaining conditions. The separability condition, which measures linearity of the core function, works counter to the minimization criterion. In each particular application this trade must be considered, but by using different values for the constant K_1, below, complete control over this trade is obtained.

The question of finding the appropriate core function can now be stated as an integer optimization problem.

A practical core can be found by selecting integers $w_1, \ldots, w_n$, which minimize the functional

$$\left[\frac{M-1}{M} C(M) + P_{max} \right] + \left[-P_{min}, \right]$$

subject to one or more of the following linear conditions as dictated by the intended application.

(i) $\Sigma\, w_i \hat{m}_i \geq 1,$

(ii) $\Sigma\, w_i$ *is odd,*

(iii) $(w_i, m_i) = 1$ *for selected* i, $1 \leq i \leq n.$

(iv) $C(M) - P_{max} + P_{min} \geq K_1 C(M),$

(v) $max\left(\sum_{i \epsilon I_+} w_i, \sum_{i \epsilon I_-} (-w_i) \right) \leq K_2.$

algorithm, each q_i is a better estimate of $[a/b]$ than the corresponding q_i of the first, but the sequence $\{q_i\}$ may alternate about $[a/b]$. Upon convergence, $q = [a/b]$ or $q = [a/b] + 1$; in either case $|a/b - q| < 1$.

Algorithm for Division II. Let $a, b \in [0, M)$, and let $q_0 = 0, r_0 = a, \delta_0 = 1$, and $s_0 = 0$.

Step (i) Verify $b \neq 0$.

Step (ii) Compare r_i and b. If $r_i < b$, then q_i is the quotient.

Step (iii) Let

$$s_{i+1} = S_{L/2}(S_L(r_i)\cdot \; b). \left[\frac{C(S_L(r_i)\cdot \; r_i}{C(S_{L/2}(S_L(r_i)\cdot \; b)\cdot \; S_L(r_i)\cdot \; b)} \right]_{rounded}$$

$$q_{i+1} = q_i + \delta_i s_{i+1}$$

$$r_{i+1} = |a - bq_{i+1}|$$

$$\delta_{i+1} = sgn\,(a-b\,q_{i+1})$$

It is clear from the earlier remark (viz., $\lim C(ka)/C(kb) = a/b$) that for sufficiently large L, this algorithm will converge. Moreover, increasing the redundant modulus will cause L to increase and therefore increase the rate of convergence. Finally, for a given L, the algorithm will converge faster for approximately linear cores than for highly nonlinear ones. It is the authors' experience that for cores which are sufficiently linear for other applications (e.g., comparison), the algorithm converges quickly.

In both division algorithms, the scaling functions S can be implemented by iterated table lookup, and the quotient of cores for the second algorithm can be implemented by table lookup.

We next discuss the computational aspects of implementing Algorithm II in a redundant RNS. We assume $5 \cdot \Delta P < C(M)$ so that the first algorithm for comparison can be used (cf. Section VI). The integers $a = (a_1,\ldots,a_{n+1})$, $b = (\beta_1,\ldots,\beta_{n+1}) \in [0,M)$ are given by their residue representations. For step (i), it must be checked that not all components $\beta_1,\ldots\beta_n$ are zero.

For step (ii), we are given $r_i = (\eta_1,\ldots,\eta_{n+1})$. The cores $C(r_i)$ and $C(b_i)$ are computed by the redundant modulus method. These values can be input to a table lookup, which signals one of the following conditions: (a) $C(r_i) - C(b) \geq \Delta P$; (b) $C(b) - C(r_i) \geq \Delta P$, or (c) $|C(r_i) - C(b)| < \Delta P$. If (a) holds, proceed to step (iii) of the algorithm. If (b) holds, the algorithm terminates. If (c) holds, $r_i - b = (\eta_i - \beta_i)$ and its core are computed. This core can be input to a table, and the decision to continue or terminate results.

For step (iii), we are given $q_i = (\rho_i)$ and $\delta_i = \pm 1$. The core $C(r_i)$ is first computed and input to a table lookup giving $s_L(r_i)$, and the residue product $s_L(r_i) \cdot r_i$ is formed. Then $C(s_L(r_i) \cdot r_i)$ is found and input to a table giving $s_L(s_L(r_i) \cdot r_i)$. If this equals 1, then $S_L(r_i) = s_L(r_i)$; otherwise the procedure is iterated to find $S_L(r_i)$. Generally, only one or two iterations are needed. Next, the residue product $S_L(r_i) \cdot b$ is formed, and $S_{L/2}(S_L(r_i) \cdot b)$ is formed as above. Note that in the iterative process for evaluating the functions S_L and $S_{L/2}$, the cores $C(S_L(r_i) \cdot r_i)$ and $C(S_{L/2}(S_L(r_i) \cdot b))$ have already been computed, being required for termination of the iterative evaluation. Since these cores are small, their rounded quotient (in residue form) can be found by table lookup. This quotient is then multiplied by $S_{L/2}(S_L(r_i) \cdot b)$ to give $s_{i+1} = (\sigma_i)$.

If $\delta_i = 1$, $\delta_i s_{i+1} = (\sigma_i)$, and if $\delta_i = -1$, $\delta_i s_{i+1} = (m_i - \sigma_i)$, implemented as a table lookup. The new quotient estimate can next be computed as a residue subtraction.

Next, $a - bq_{i+1}$ is evaluated and its core computed via the redundant modulus method. Its core is also computed by the second corollary to Theorem 2. If these results are equal, $a - bq_{i+1} \geq 0$, otherwise they will differ by C(M) and $a - bq_{i+1} < 0$. This gives δ_{i+1}, and r_{i+1} is found by componentwise m_i-complementing if necessary.

Example 8. Estimate a/b = 1000000/3456 using the moduli set {23,25,27,29,31}, core weights {-3,-2,4,-1,3}, and redundant modulus $m_6 = 256$. System constants are $P_{min} = -6.76$, $P_{max} = 5.76$, $\Delta P \approx 12.51$, $L = 50{,}722{,}113$. For clarity, the solution is given in decimal form.

Iteration 1. $r_o = 1{,}000{,}000\ (>b)$.

$S_L(r_o) = 38$

$S_L(r_o) \cdot r_o = 38{,}000{,}000$ with core 183

$S_L(r_o) \cdot b = 131{,}328$

$S_{L/2}(S_L(r_o) \cdot b) = 168$

$S_{L/2}(S_L(r_o) \cdot b) \cdot S_L(r_o) \cdot b = 22{,}063{,}104$ with core 108

quotient of cores = 2

$s_1 = q_1 = 336$

$\delta_1 = -1$

$r_1 = 161{,}216\ (>b)$

Iteration 2. $S_L(r_1) = 204$

$S_L(r_1) \cdot r_1 = 32{,}888{,}064$ with core 159

$S_L(r_1) \cdot b = 705024$

$S_{L/2}(S_L(r_1) \cdot b) = 22$

$S_{L/2}(S_L(r_1) \cdot b) \cdot S_L(r_1) \cdot b = 15{,}510{,}528$ with core 76

quotient of cores = 2

$s_2 = 44$

$q_2 = 292$

$\delta_2 = -1$

$r_2 = 9512\ (>b)$

Iteration 3. $S_L(r_2) = 3906$

$S_L(r_2) \cdot r_2 = 35{,}747{,}712$ with core 173

$S_L(r_2) \cdot b = 13{,}499{,}136$

$S_{L/2}(S_L(r_2) \cdot b) = 1$

$S_{L/2}(S_L(r_2) \cdot b) \cdot S_L(r_2) \cdot b = 13{,}499{,}136$ with core 69

quotient of cores = 3

$s_3 = 3$

$q_3 = 289$

$\delta_3 = 1$

$r_3 = 1216$

Since $r_3 < b$, $a/b \approx q_3 = 289$. To 4 decimal digits, $a/b = 289.3519$.

Statistics

Three simulation runs of 200 trials each were done to test the division algorithm in the RNS with moduli set {23, 25, 27, 29, 31}. In each trial two numbers, *a* and *b*, were generated. The first one was generated at random from the complete range of the RNS. The second number was generated at random from the range *[0,a)* for RUN1, the range *[0,a/1000)* for RUN2, and the range *[0,a/1000000)* for RUN3. In each trial two sets of data were recorded, the absolute error between the true quotient *a/b* and the quotient evaluated by the algorithm, and the number of iterations required by the algorithm. The two tables below give some summary statistics obtained from the simulations.

STATISTICS FOR ABSOLUTE ERROR	RUN1	RUN2	RUN3
minimum	3.7E-4	3.1E-3	0
25 percentile	.123	.176	0
50 percentile	.253	.301	0
75 percentile	.395	.534	.333
maximum	.901	.995	.889

STATISTICS FOR NUMBER OF ITERATIONS	RUN1	RUN2	RUN3
minimum	1	2	3
25 percentile	1	3	6
50 percentile	1	4	6
75 percentile	1	4	7
maximum	4	6	9

A similar set of simulations was performed using the first division algorithm. As with the present algorithm, the absolute errors never exceeded one, but had a higher incidence of errors above one half. The statistics on the number of iterations show that the first algorithm has very slow rate of convergence, viz., 30% of the trials required over 20 iterations.

VIII. CODING METHODS

The purpose of this hardware study was to investigate the feasibility of producing a collection of VLSI chips which might be used to implement various digital signal processing algorithms. A chip set would be made up of the basic arithmetic functions, as well as some special functions. An important special function is the operation of calculating the core. The design and fabrication of a core calculator chip was the only way that true feasibility could be demonstrated. This effort was made possible by the advent of new VLSI computer aided design technology. With the aid of an advanced silicon compiler, it was possible to exercise several design options in a relatively short period of time and to implement the final design, all with relatively low cost. The major design considerations were: multiple operations per chip, arithmetic speed, the ability to use relatively large moduli, silicon area, and chip delay.

In the past, nearly all RNS implmentations have used read only memories (ROM's) for the basic arithmetic in a binary coded lookup table format [Huang, 1981; Jenkins, 1977; Jullien, 1980; Polky, 1982; Soderstrand, 1981]. The hardware elements have also been commercially available components of either TTL, ECL, or NMOS technology. A typical TTL ROM would be organized as a 1K-4K by 8 bit memory, where a single chip package would perform each arithmetic operation. These devices typically execute their operations on the order of 20-40 nsec (TTL). The propagation delay can be further reduced by using ECL RAM's (7nsec), however the package count increases by eight times since these devices are only organized as 1K by 1-bits [Huang, 1980].

VLSI devices fall into two typical categories, single operations with special feature or more complex devices to be used as building blocks in digitial signal processing algorithms. For example, Taylor [1982] conceptually designed a multiplier device for large (48-72 bits) dynamic range operations, which was composed of a modulo $2^n + 1$ adder and a PLA. Other conceptual designs have taken a similar approach [Jenkins, 1982] for general purpose adders, subtractors, and multipliers, all using a binary adder and a ROM. Jenkins suggests that in addition to the basic operations, a standard device could be implemented to perform a mixed radix conversion kernel, a nested polynomial kernel, and complex arithmetic. More recently, Jullien [1983] has proposed an all memory structure using multi-look-up table modules for digital filter applications. He discusses the chip area and delay time for various layouts.

An example of a single function 8-bit RNS multiplier has been presented by Ching and Johnsson [1983] which was submitted for fabrication by the DARPA MOSIS foundry service. They concluded that a lookup table format is preferable for moduli coded with less than 4-bits, while an array multiplier architecture would be better for moduli greater than 5-bits. Their analysis considered both 4μm and 0.5μm feature sizes, with the final design implemented in 4μm NMOS. Another more complex chip design was realized by Yeh et.al [1983], where they implemented a 32-point Fermat number transform for FIR filters.

In all current cases of VLSI implemented RNS devices, the designs have dealt with either creating a real-time alternate for a time consuming single functions, such as large dynamic range multiplication; or with some special purpose function such as the number theoretic transform. Our work has attempted to understand better the potential for VLSI implementation of more general purpose operation which would be applied specifically to digital signal processing problems. The core calculator chip would be the foundation of a building block collection of RNS devices. These components could then be used with the aid of a computer aided design software package to develop system level modules.

Several coding methods were evaluated with regard to VLSI implementation and their effects on silicon area and propagation delay through RNS computational devices. Two general classes of codes for residue representation are redundant and nonredundant codes [Szabo, 1967]. It is assumed that residue codes for any integrated circuit devices will be of binary nature however with relationships between bits depending on the specific type of code. For this investigation the nonredundant code was selected to be simply as fixed weight binary code commonly used in all computer systems. The redundant codes were of two types: 1-of-m position code, and 2-of-m position code for a modulus of m.

The 1-of-m code is made up of m binary bits with m-1 zeroes and only a single 1 in the kth position of the word. Residue operation with this kind of code occurs simply as a permutation of the position of the nonzero bit in the output codes based on the respective input code positions. In terms of memory usage, this type of code is inefficient because only m of the 2^m binary states are used. However, it was thought to have the potential for requiring less control logic than other, more compact codes.

The 2-of-m code differs in that two bits are non-zero rather than one. The primary advantage of this code is that it requires fewer bits to represent the set of integers in any residue base. The 2-of-m position code is constructed by using the m-1 left-most bits to represent half the number range, with the right-most bit indicating which half. Consider the following example, for modulo 7 arithmetic:

digit	code
0	0000
1	0010
2	0100
3	1000
4	1001
5	0101
6	0011

The sets [1,2,3] and [4,5,6] are similar in that their three left-most coded bits have only one non-zero element, whose position determines the difference between individual numbers. Another advantage of this code is simple determination of the additive inverse. In each case the additive inverse corresponds to the number which has the same most left-most bit positions (e.g. 1 → [001] and 6 → [001]). The 2-of-m code has properties similar to a conventional 2's complement code, where the most significant bit is use to determine whether a number is positive or negative. An example would be +3 → [0011] and -3 → [1101]. A disadvantage of the 2-of-m code is that its representation for different moduli is not the same. For example, the integers 4 mod 5 and 4 mod 7 are represented by 0101 and 1100 respectively.

A modulo 7 adder was selected as a model device for comparing these three codes and each programable logic array (PLA). A PLA represents a compact arrangement of primitive gates which can accomodate a variety of complex logical operations with efficient use of silicon area. Figure 4 illustrates the geometric layout of a typical PLA. The cell is composed of rows and intersecting columns for both the AND array and the OR array, where the rows also connect the two arrays together. The number of columns is related to the number of input/output terms in the logic function. A standard PLA cell is composed of an AND matrix and an OR matrix, which are linked side by side, with the inputs going to the AND array and output leaving the OR array. A PLA implements a canonical set of sum of products combinatorial logic functions of n-inputs and m-outputs. The logical structure is similar to that of a read only memory (ROM), except that it does not cause as many

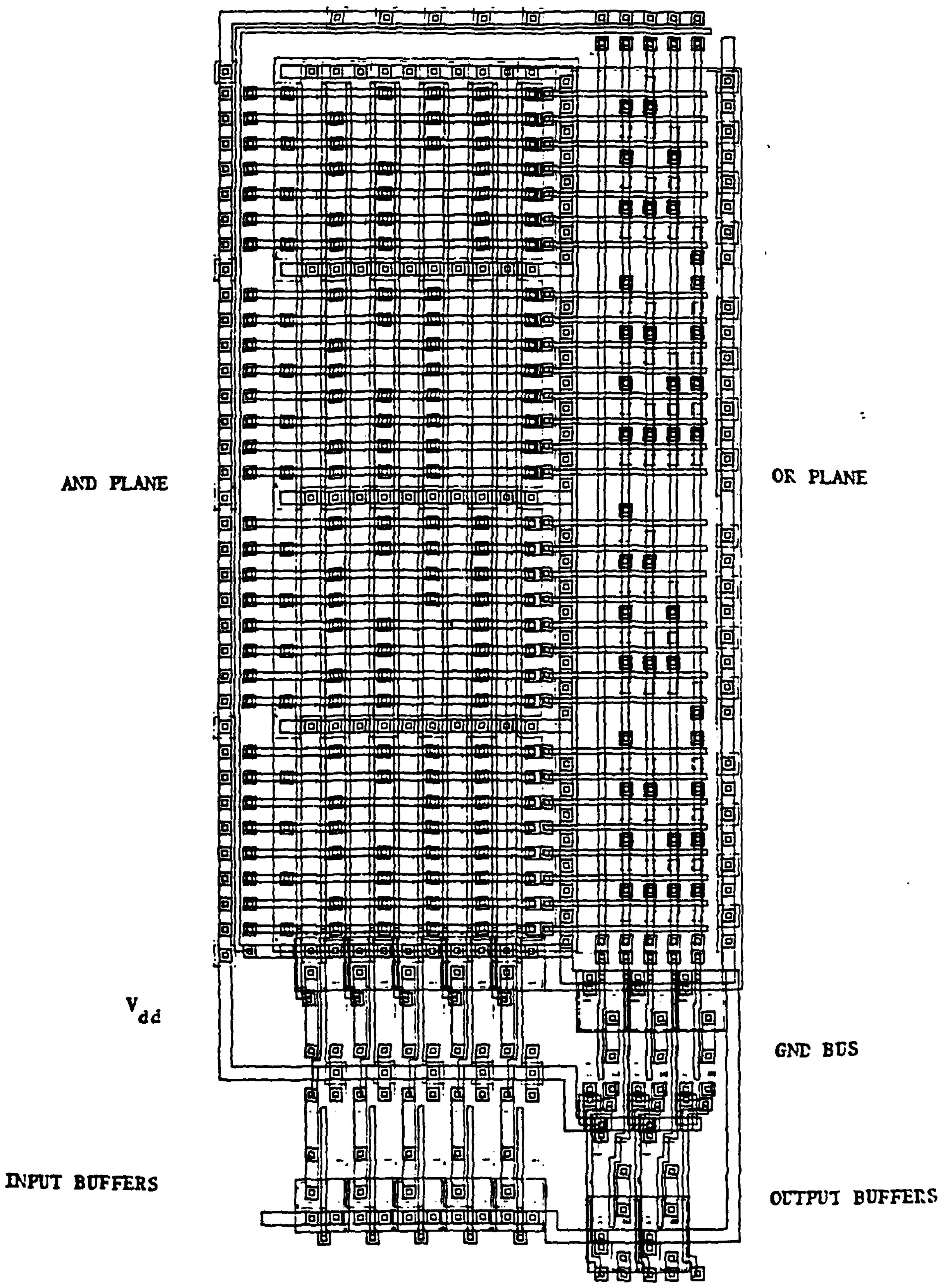


Fig. 4. Layout drawing for programmable logic array (PLA).

unused states as a standard read only memory ROM lookup table might. This occurs because a ROM usually has a preprogrammed AND array which allow address inputs to logically select a single m-bit data value.

The Seattle Silicon Concorde^{tm} silicon compiler was used to generate the PLA structures for the three coding techniques. High level logic equations were created for each type of addition operation and were then optimized by a software logic reduction routine to produce the canonical sum of products equations. Figure 5 illustrates the logic reduction and optimization for a mod 7 adder, where the original statement of the adder function is based on a truth table. The truth table entries are in radix 10, while the final logic equations are for a binary coded residue system. In the figure, the operands are A and B with C being the result. Logical operations of AND and OR are symbolically represented by & and # respectively. The logical complement is indicated by the ! symbol. This sum of products form of logic equations is easily mapped into the AND/OR physical structure a PLA module. The reduced logic equations were entered into the compiler's input stream for further processing. Based on the corresponding minterms for each equation set, the compiler generated the PLA cells. A graphical display monitor was used to investigate the geometric nature of each PLA and to determine the amount of silicon area which would be occupied by each cell. The compiler also generated additional information needed for gate level simulations and timing.

The resulting silicon areas for the three PLA structures is summarized in Table I. These results indicate that a binary coded device would use the least number of inputs, outputs and columns. The significance of the number of columns is that it will influence the propagation delay between the two arrays. The small number of input/output terms is due to the compact nature of conventional binary coding. However, it was unexpected that the silicon area and the number of PLA columns are also less than other code implementations.

The propagation delay was determined by evaluating the addition of 10 random numbers in each PLA. The average delays are given in Table II. Again, the binary code exhibited the best performance, which is consistent with the general rule that cells with the smallest area have the smallest delays. It is estimated from this analysis that for moduli larger that 7, the binary code will continue to be the most efficient implementation method for VLSI RNS components.

Table I

Sizing of PLA structures for three coding methods

	1 of m	2 of m	binary
number of input/output	21	12	9
number of columns	49	51	36
area (sq. mil)	943	630	384

Table II

Average propagation delay for the addition of ten numbers.

	1 of m	2 of m	binary
propagation time (nanoseconds)	21.8	15.4	13.5

```
(a)  truth_table([a,b]->c)
                    [0,0]->0;     [1,0]->1;     [2,0]->2;
                    [0,1]->1;     [1,1]->2;     [2,1]->3;
                    [0,2]->2;     [1,2]->3;     [2,2]->4;
                    [0,3]->3;     [1,3]->4;     [2,3]->5;
                    [0,4]->4;     [1,4]->5;     [2,4]->6;
                    [0,5]->5;     [1,5]->6;     [2,5]->0;
                    [0,6]->6;     [1,6]->0;     [2,6]->1;


                    [3,0]->3;     [4,0]->4;     [5,0]->5;
                    [3,1]->4;     [4,1]->5;     [5,1]->6;
                    [3,2]->5;     [4,2]->6;     [5,2]->0;
                    [3,3]->6;     [4,3]->0;     [5,3]->1;
                    [3,4]->0;     [4,4]->1;     [5,4]->2;
                    [3,5]->1;     [4,5]->2;     [5,5]->3;
                    [3,6]->2;     [4,6]->3;


                    [6,0]->6;
                    [6,1]->0;
                    [6,2]->1;
                    [6,3]->2;
                    [6,4]->3;
                    [6,5]->4;
                    [6,6]->5;
```

Fig. 5. Logic reduction for a mod 7 adder showing (a) the truth-table source and (b) the reduced logic equations for a PLA device.

(b) Reduced Equations:

```
c2 = (!b2 & b0 & !a2 & a1 & a0
     # (b2 & b1 & !b0 & a2 & !a1 & a0
     # (!b2 & b1 & b0 & !a2 & a0
     # (b2 & !b1 & b0 & a2 & a1 & !a0
     # (b2 & b1 & !b0 & a2 & a1 & !a0
     # (!b2 & !b1 & !b0 & a2 & !a0
     # (!b2 & b1 & !a2 & a1
     # (b2 & !b1 & !b0 & !a2 & !a0
     # (!b2 & !b0 & a2 & !a1 & !a0
     # (!b2 & !b1 & a2 & !a1
     # (b2 & !b0 & !a2 & !a1 & !a0
     # b2 & !b1 & !a2 & !a1)))))))))));
```

```
c1 = (b2 & b1 & !b0 & !a2 & a1 & a0
     # (b2 & !b1 & a2 & !a1 & a0
     # (!b1 & b0 & !a1 & a0
     # (!b2 & b1 & b0 & a2 & a1 & !a0
     # (!b1 & !b0 & a1 & !a0
     # (!b2 & !b1 & !a2 & a1 & !a0
     # (!b2 & !b1 & !b0 & !a2 & a1
     # (b2 & !b1 & b0 & a2 & !a1
     # (b1 & !b0 & !a1 & !a0
     # (!b2 & b1 & !a2 & !a1 & !a0
     # (!b2 & b1 & !b0 & !a2 & !a1
     # !b2 & b1 & b0 & !a2 & a1 & a0)))))))))));
```

```
c0 = (b2 & !b1 & b0 & !a2 & a1 & a0
     # (!b2 & b1 & b0 & a2 & !a1 & a0
     # (!b2 & !b0 & !a2 & a0
     # (b1 & !b0 & a2 & a1 & !a0
     # (b2 & b1 & !b0 & a1 & !a0
     # (b2 & !b0 & a2 & !a0
     # (!b2 & b0 & !a2 & !a0
     # (b2 & !b1 & b0 & a2 & !a1 & a0
     # (!b2 & !b1 & !b0 & !a1 & a0
     # (!b1 & !b0 & !a2 & !a1 & a0
     # (!b2 & !b1 & b0 & !a1 & !a0
     # !b1 & b0 & !a2 & !a1 & !a0)))))))))));
```

Fig. 5 (cont.)

an of VLSI Chip

angement of cells in the core chip corresponds closely with the arithmetic elements of the nction, fundamentally an inner product. PLA cells were used to affect the multiplication on, while 5-bit binary adders were used for a chain summation. In RNS terms the chip s a residue representation $a = (a_1, a_2, a_3, a_4, a_5, a_6)$, where the moduli set is ,25,27,29,31,32}. The chip will execute the function,

$$C(a) = \left| \sum s_i a_i \right|_{m_{n+1}} ,$$

the terms s_i are the fixed coefficients {7,24,6,20,30,9}.

determined that a modulus of 32 would be an acceptable choice not only for response related core function, but by selecting a power of two, the basic arithmetic elements on the chip could lt from conventional 5-bit binary adders and PLA's. . Five bit binary adders were selected for d 32 operations, while PLA's were selected to perform the fixed coefficient multiplies. Figure trates the core chip floor plan. Six residue values enter the chip through PLA's wired to m the set of fixed multiplications, followed by binary adders which accumulate the totals by ummation (mod 32), with the resulting core value produced at the chip outputs .

Design

core calculator chip design two standard compiler components were used: the PLA and the inary adder. The PLA's are programmed by their geometric interconnections to act essentially oders. In more conventional terms, the residue values represent a code which is used to select M possible output states. For example, the mod 23 PLA would produce the following results:

input = 5	output = 3 mod 32
input = 9	output = 31 mod 32.

$b_{i-1}|_M$

$$\begin{cases} 1 & \text{if } 2.b_{i-1} > M, \\ 0 & \text{otherwise,} \end{cases}$$

$_1 + \varepsilon_i a_i$.

multiplication. Delta multiplication assumes that the RNS can be subdivided into two
tems with ranges M_1 and M_2, where $M = M_1.M_2$, and $M_1 \approx M_2 \approx \sqrt{M}$. The value of c is
ted by the product of the integer parts of a/M_1 and b/M_2, and adjusted by the residues of a
modulo M_1 and M_2, respectively.

$= [a/M_1], b^* = [b/M_2], r(a) = |a|_{M1}, r(b) = |b|_{M2}$. Then $a = a^*.M_1 + r(a), b = b^*.M_2 + r(b)$
$c' = (a.b/M) = a^*.b^* + \{b^*.r(a)/M_1\} + \{a^*.r(b)/M_2\} + \{r(a).r(b)/M\}$. In the last expression
oduct $a^*.b^*$ is directly computable in the RNS since $a^* < M_2$ and $b^* < M_1$. The next two terms
e divisions by M_1 and M_2 respectively. Defining $\Delta(a) = [r(a).b^* / M_1]$, $\Delta(b) = [r(b).a^* / M_2]$,
$|r(a).b^*|_{M_1}$, $s(b) = |r(b).a^*|_{M_2}$, it follows that $c' = a^*.b^* + \Delta(a) + \Delta(b) + d$, where

$$d = \frac{r(a).r(b)}{M} + \frac{s(a)}{M_1} + \frac{s(b)}{M_2}.$$

lue of $c = [c']$ is evaluated as $a^*.b^* + \Delta(a) + \Delta(b)$ with an error which does not exceed 3.

sidue representations of $r(a)$ and a^* are obtained in two steps. First the representations
the subsystem of range M_1 are evaluated. The complete residue representations are then
ed by using the extension of base theorem. The subsystem representation of $r(a)$ follows
iately since it is identical to that of a. That of a^* is obtained by dividing a-$r(a)$ by M_1 in the

ɜnotes the product *a.b (mod M)*, then $a.b = c.M + e$, and $C(a.b) = c.C(M) + C(e)$. If $C(M) \neq 0$, ɹsing (A.1), it follows that

$$c = \{ b.C(a) + \Sigma\, w_i \alpha_i (b - \beta_i / m_i) + \Sigma\, w_i [\alpha_i \beta_i / m_i] - C(e) \} \div C(M)$$

alue of c can then be evaluated exactly in the RNS if M and $C(M)$ are relatively prime. This ithm is exact, nevertheless for a system with n moduli it requires several core evaluations, n gs (Theorem 9), and n table look-up operations.

y Multiplication. The binary algorithm uses the binary expansion of the quotient *b/M*, and ates c as

$$c \approx \sum_{i=1}^{k} \varepsilon_i [a/2^i], \qquad \text{(A.2)}$$

e $k = [\log_2 (M-1)]$, and the bits ε_i are the coefficients in the binary expansion of *b/M*. The error ng (1) to evaluate c is

$$\sum_{i=1}^{\infty} \varepsilon_i \frac{|a|_{2^i}}{2^i}$$

ı is bounded above by $k - 1 + (M/2^k)$. The implementation of this algorithm consists of the ıation of $[a/2^i]$; the evaluation of the bits ε_i, and the sum in the right hand side of (A.2). The first ·equires at most k parity checks and scalings by 2. The binary expansion of *b/M* is attained by ıarisons of M with the product of b with successive powers of 2. The algorithm starts with l values $a_0 = a$, $b_0 = b, \varepsilon_0 = c_0 = 0$, and then uses the recursive formulas given below. The process ınates when $a_i = 0$, which will occur for some $i \leq k$.

parity of a_{i-1}

$[a_{i-1}/2] = (a_{i-1} - d_i)/2$

PENDIX. FLOATING POINT ARITHMETIC

s section presents the application of the notion of core function to mantissa operations in floating ınt arithmetic. Akushskii et al [1], [2] discuss three multiplication algorithms and one division ıorithm: core, binary, and delta multiplication, and binary division. All four algorithms use the 'e function, but whereas core multiplication is based on a formula for the core of a product of two egers, the other algorithms use the core indirectly to perform comparisons, extensions of base, d scalings.

floating point arithmetic, each number is represented by an exponent and a mantissa. The ponent, an integer, can be represented in the usual way as a power of 2. The mantissa, a number tween 0 and 1, is represented by the element of the RNS whose quotient to the range of the ıtem is equal to the mantissa. More precisely, a number $x.2^k$, $0 \le x < 1$, is represented by the pair $k)$ where $0 \le a < M$, and $[x.M] = a$.

t us consider two integers a and b in $[0,M)$, which represent the mantissas a/M and b/M spectively. The product mantissa $(c'/M) = (a/M).(b/M)$ is represented by the integer part $c = [c'] =$ $b/M]$. A general description of each of the algorithms used for evaluating the product c are given low. All three approaches are then applied to an example, which show in detail how each gorithm is implemented in residue arithmetic with core functions. Since the same example is used, is permits us to compare the algorithms in terms of complexity and speed.

re Multiplication. The core multiplication algorithm is based on a formula for the core of a oduct of two integers:

$$\begin{aligned} C(a.b) &= b.C(a) + \Sigma\, w_i [\, b.\alpha_i / m_i] \\ &= b.C(a) + \Sigma\, w_i \alpha_i (b - \beta_i / m_i) \qquad \text{(A.1)} \\ &\quad + \Sigma\, w_i [\alpha_i \beta_i / m_i]. \end{aligned}$$

ͽ made available for laboratory evaluation. After the performance limitation of these devices is etermined, the resulting insights will be applied for further evaluation of speed enhancements, naller line widths (e.g., 1μm), and more input/output pins.

ı addition to the specific research discussed above, a more general effort will be pursued to ıvestigate similar design techniques applied to devices for division and other floating point perations. A computer-aided engineering system will be used to simulate the performance of arious RNS devices. Software simulation tools will allow a hierarchical approach to the overall esearch effort and thereby provide greater insight into the high level functional behavior of ıotential subsystems for a real-time signal processing system.

ıpeed and hardware requirements should be evaluated for specific architectures using cores and/or he fractional RNS for realistic problems. Simulations of critical subprocesses using a computer aided lesign engineering workstation should be performed. Specific algorithms of interest to the Ocean ;urveillance Signal Processing program should be determined as benchmark algorithms for the ;valuation process. Given an algorithm, an RNS computational architecture could then be letermined and the functional layout of the integrated circuits and the data flow paths would be pecified. Execution times and hardware requirements could then be evaluated and compared with hose predicted using a more conventional architecture.

The current algorithms of Gregory and Matula are computationally complex and require high-precision calculations in a weighted number system, in part defeating the primary benefit of using an RNS. Decoding procedures performed within a usual (not fractional) RNS itself will be investigated. One approach would be to recast Gregory and Matula's algorithm into the RNS; however, the difficulty here is requirement for integer division in repeated applications of the Euclidian algorithm. If a core could be derived which is "nearly linear", then the integer divide might be efficiently performed using the core to make successively more accurate estimates of the quotient.

Gregory and Matula have also developed a yet unpublished alternate fractional RNS. Instead of using n-tuples of ordered pairs to represent a fraction in an n-modulus RNS, n-tuples of integers are used together with a formal symbol. The formal symbol will appear in the k^{th} position of the representation of a fraction if the denominator is divisible by the k^{th} modulus. Arithmetic operations are then defined on the residues together with this formal symbol. This system will be examined in detail for application to signal processing problems, and new algorithms to alleviate the decoding bottleneck will also be investigated.

Having developed more efficient approaches for the operations of decoding and magnitude comparison, the computational complexity of the fractional RNS on realistic problems will be evaluated. Comparisons with other RNS and more conventional architectures will be performed.

Research should continue toward investigation of VLSI techniques applied to special purpose RNS devices. Further analysis for functional performance and execution speed of a core calculation chip is required. These efforts could be performed with the aid of a silicon compiler CAD system. By using such a method, considerable insight can be gained regarding the detailed physical layout and logical performance of a conceptual RNS processor chip. By taking advantage of a low cost shared-chip fabrication process, a small number of devices can be produced for further research of real chip performance. It is expected that both a core calculation chip and a multi-modulus adder chip would

Many other applications of the core function may exist, and new and even simpler algorithms for the difficult residue operations may be possible. First, analysis of convergence properties of the new division algorithms should be performed. It is expected that further refinements of the algorithms will yield better convergence properties. Second, new overflow detection methods for both addition and multiplication are probably possible for suitably linear core functions. Next, analysis of appropriate integer programming techniques for computation of optimal core functions is required. Improvement of the floating point division and multiplication algorithms of Akushskii et al. should be investigated. Possibly, core functions could be utilized to attain a unique representation of a floating point number. Next, the analysis of the Soviet RNS literature for this project has been so fruitful that further efforts in this direction are required. Finally, a variety of other uses for the core will undoubtably arise; for example core functions may be applied to fault tolerant systems.

In the course of the current project, the fractional RNS concepts of Gregory et al. [11] were surveyed for their application to real time signal processing. In a fractional RNS a mapping is defined from a subset of the rationals to a residue system which permits divisions as fast as multiplications and additions. The work was developed for exact arithmetic in poorly conditioned linear algebra computations. Consequently, it was implemented on a general purpose computer using extremely large moduli and an extremely large dynamic range (on the order of hundreds of bits). This work is not only mathematically elegant, but it may have great potential for the smaller dynamic range problems found in signal processing. In this setting, encoding into the system and all the elementary arithmetic operations (including division) are efficient and can be implemented in a single clock cycle using custom or semi-custom hardware. The operations of magnitude comparison and decoding back to the fractional representation are currently quite difficult, and the development of new algorithms for these operations is expected to permit the application of the fractional RNS to complex signal processing problems.

IX. CONCLUSION AND FUTURE WORK

This paper has presented the core function of Akushskii, Burcev, and Pak and has extended their theoretical results to render the core function a practical solution to the difficulties of residue arithmetic. Specifically, this paper has introduced the use of a redundant modulus for efficient core calculation and has presented algorithms for all of the traditionally difficult residue operations including comparison and general division. In addition, new results were presented which reformulate the problem of selecting the core coefficients w_i as an integer programming problem, so that optimal cores may be obtained. High speed VLSI circuits for implementing the core function and other multi-stage RNS computations have been judged feasible.

In the past, the residue number system has been well matched to the performance of linear signal processing algorithms for which very-high speed execution was critical. With the results presented in this work, it now appears that the RNS may be suited for high speed performance of nonlinear arithmetic and complex logical operations. These efforts have been very positive, both in the solutions of some theoretical problems and in leading to physically realizable RNS core calculation integrated circuits. By coupling the parallelism of the RNS with the positional capabilities of the core function, RNS computational architectures offer high potential for next generation processors for both linear and non-linear real time processing.

Further research is required before this potential can be realized for practical data processing systems. Four areas have been identified for continued investigation. These include (i) further analysis of the applications of the core function; (ii) investigation of fractional RNS systems; (iii) conceptual design and analysis of RNS VLSI devices; and (iv) development to facilitate applications of this technology to signal processing.

moduli	a_i	s_i	product (mod 32)
23	10	7	6
25	20	24	0
27	5	6	30
29	25	20	20
31	19	30	26
32	30	9	14
		core (mod 32) =	0

Based on the simulation with the above stated test numbers, the propogation delay for the entire chip was found to be 60 nsec. This is a respectable performance for a multiple operation RNS calculation. As a comparison, if single commercially available chips were used, then the delay might be 6X35nsec = 210 nsec (assuming one multiply and 5 adds). Also it is possible that further optimization could be achieved for the VLSI layout, so that a delay shorter than 60 nsec might be attained.

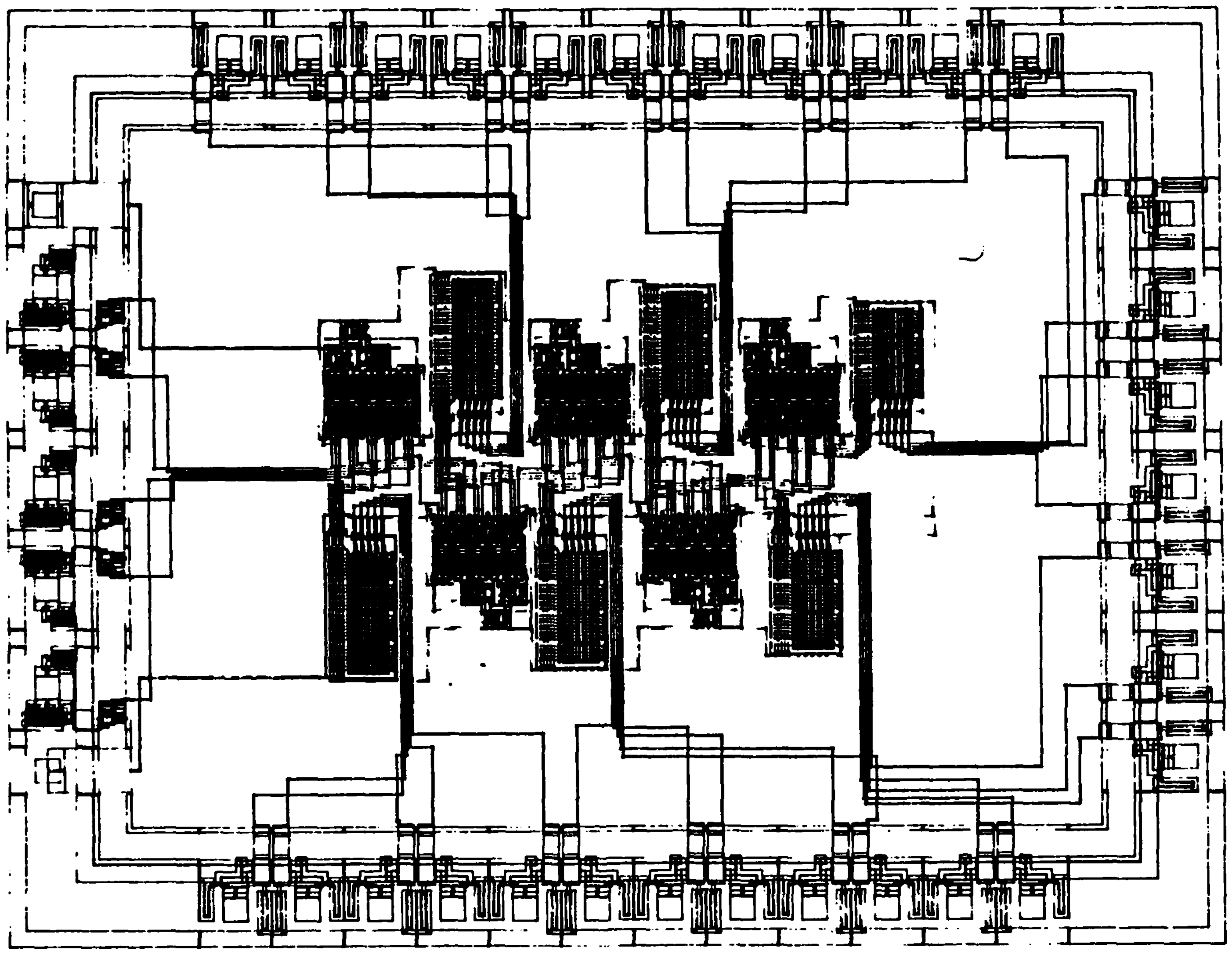


Fig. 9. The complete layout drawing for the core calculator chip showing PLA cells, binary adder cells and the pad ring.

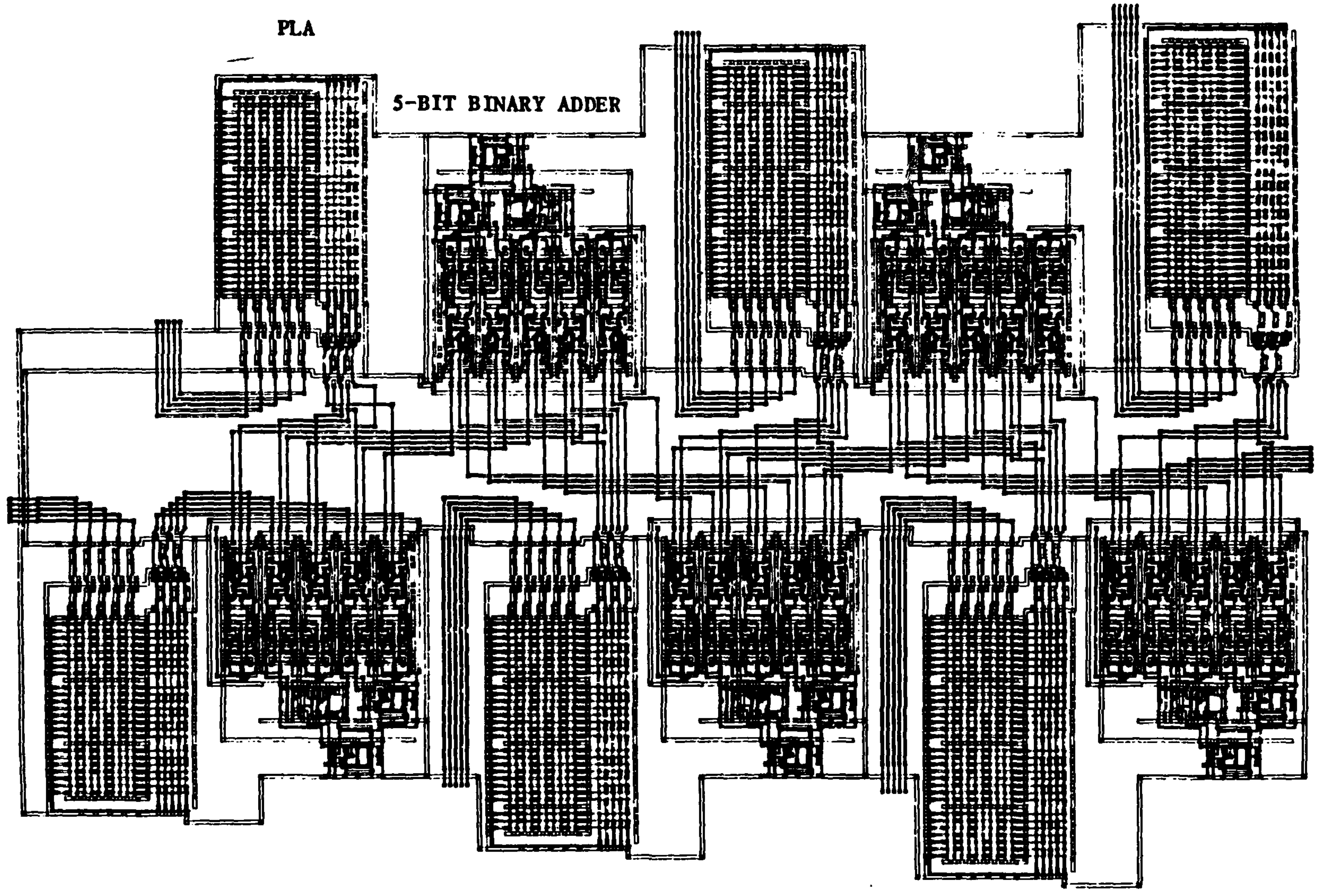


Fig. 8. PLA/binary adder cells connected to calculate the core function mod 32.

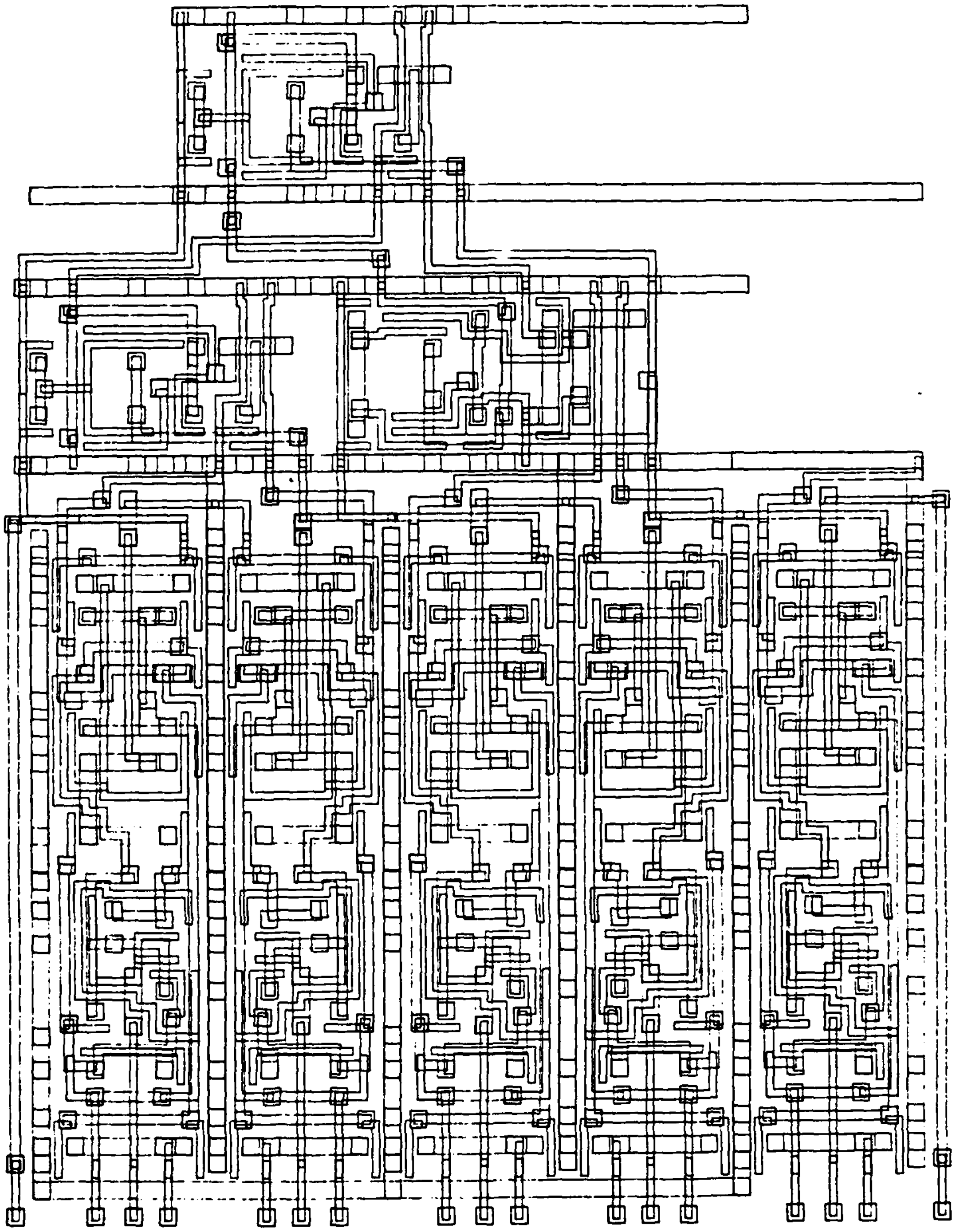


Fig. 7. Layout drawing for a 5-bit adder cell.

The PLA outputs feed five 5-bit binary adders, whose outputs are then connected to the inputs of the next stage. Since the addition of all arithmetic is done mod 32, this chained arrangement facilitates data flow. Figure 7 represents the silicon design for the adders, where the nature of the geometric patterns is similar to the previously described PLA cell.

Figure 8 illustrates the working area of the core calculator, whose dimension is 70 x 100 mils. (This is smaller than a single cell for a 1-of-m or 2-of-m position code). Figure 9 shows the complete chip, including an input/output pad ring, 30 inputs, 5 outputs and one power/ground pair, with a total area of 155 x 200 mils.

Simulation

The core calculator design was tested using RNL, a timing logic simulator for digital circuits, which uses a resistive model of transistors to implement a logic level simulations. This simulation completely checks the functionality of the circuitry, and has been determined accurate from previous research to be within 20 percent of the performance measured for devices in actual fabrication.

The entire core chip was evaluated by processing a set of test vectors which were selected as the numbers:

1835870 10545965 6401369 7435820 3056008 9474898

13045486 5352529 11597763.

This set was encoded by each modulus to produce the input residues for the chip input pins, and the resulting core values from the chip simulation were compared with the analytical values. A successful match between these results was used as a strong indication that the chip would function properly after fabrication. A sample calculation would be similar to the following:

The silicon representation of the standard PLA cell is shown in Fig. 4. The design shows the various layers (metal, poly silicon, diffusion, etc.) of the CMOS p-well fabrication process. As shown in the Figure , the PLA is surrounded by Vdd and GND metal traces. The inputs to the PLA occur at the five contacts located in the lower left hand corner. The input signals proceed north through buffers where the drive capacity and propagation delays of the signal can be controlled.

The signals exit the buffers and enter the plane of the PLAs , which consists of minterm rows representing the binary form of the input vector. A modulo 31 PLA has 31 rows of minterms, representing inputs of 0-30. Physically these minterms are a series of transistors that form logic gates corresponding to the PLA code file. For example, the code file for row 2 of this PLA is

0 0 0 1 0

which is the binary representation of the number 2. Silicon transistors are formed in row 2 such that if the PLA input is equal to 2, then the output will correspond to the value stored on the right hand side of row 2, which forms the OR plane of the PLA.

The PLA in Figure 4 multiplies the input signal by 30 and then outputs the product modulo 32. For the above example, an input of 2 will yield an output of: $2 \times 30 \equiv 28 \pmod{32}$.

Thus the right hand side of row 2 has transistors which represent the binary code for 28, or

1 1 1 0 0.

The result of an input signal equal to 2 produces an output signal equal to 28, which is the correct response for this PLA. The electronic input signals pass from the input contacts through the proper minterm and finally through the output buffers to the output contacts for wiring connections to other parts of the core calculator.

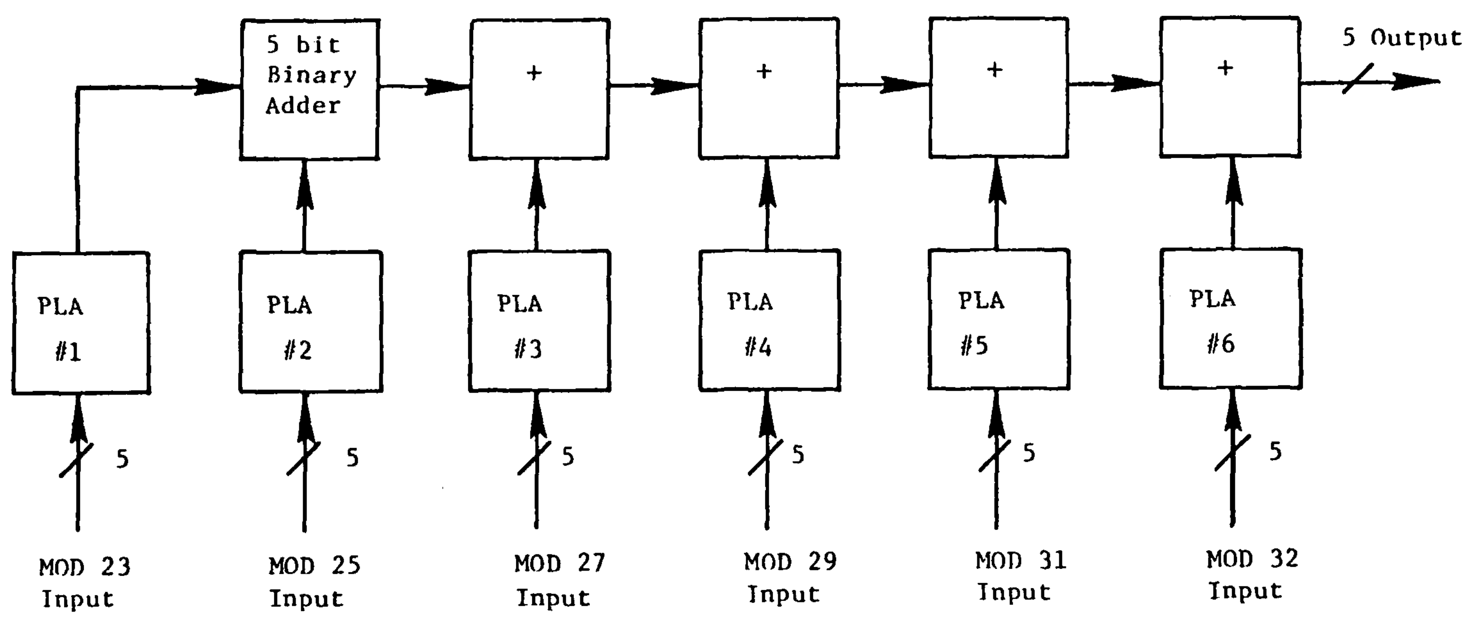


Fig. 6. Floor plan of the core calculator chip design.

subsystem of range M_2 and then applying extension of base to the subsystem M_1. Similar steps are used to evaluate the rest of the terms.

Example 9. Consider an RNS with moduli {3,5,7,11}. The core function has weights {-1,2,-1,1}, and hence $C(M) = C(1155) = 17$. The orthogonal basis elements are $B_1 = 385$, $B_2 = 231$, $B_3 = 330$ and $B_4 = 210$, with cores $C(B_1) = 6$, $C(B_2) = 3$, $C(B_3) = 5$, and $C(B_4) = 3$. The core has a minimum of -2 and a maximum of 18, and so 0,1,15, and 16 are critical cores.

In the present example the multiplicands are $a = (0,1,5,8)$ with core $C(a) = 1$, and $b = (2,4,6,9)$ with core $C(b) = 13$. The value of $c = [a.b/M]$ is evaluated using the three algorithms described above. The decoded values of the operands and quotients as well as the errors are discussed following the example.

(*i*) Core Multiplication.

When formula (*) is applied several of the terms are easily computed. First $b.C(a) = (2,4,6,9) \times 1 = (2,4,6,9)$. The third term may be computed using a table look-up to evaluate each of the summands $wi\,[a_i\,\beta_i / m_i]$, and then adding to obtain (2,2,2,2). The remainder $e = |a.b|_M = (0,4,2,6)$ has core $C(e) = 6$, and so $-C(e) = (0,4,1,5)$. The division by $C(M) = 17$ is implemented by multiplying by its reciprocal $(1/17) = (2,3,5,2)$, which is a system constant. The first sum is the only remaining term and is the most demanding. It requires one scaling of b per modulus, e.g., for the first modulus $b\text{-}b_1 = (0,2,4,7)$ with core $C(b\text{-}b_1) = 13$. The residue representation of $(b\text{-}b_1) / m_1$ in the subsystem of moduli {5,7,11} is

$$\frac{b - m_1}{m_1} = \frac{(2,4,7)}{(3,3,3)} = (2,4,7) \times (2,5,4) = (4,6,6).$$

Using Theorem 9 to scale b by m_1 it follows that the remainder of $(b\text{-}b_1) / m_1$ with respect to m_1 is 2. Repeating this process for each of the rest of the moduli one obtains $(b\text{-}b_2)/m_2 = (2,3,6,1)$, $(b\text{-}b_3)/m_3 = (2,4,1,2)$, $(b\text{-}b_4)/m_4 = (1,0,1,8)$, and so $\Sigma\, w_i a_i (b\text{-}\beta_i / m_i) = (2,1,1,1)$.

Formula (A.1) is now applied to calculate $c = \{(2,4,6,9) + (2,1,1,1) + (2,2,2,2) + (0,4,1,5)\} \times (2,3,5,2)$ $= (0,1,3,6) \times (2,3,5,2) = (0,3,1,1)$.

(ii) Binary Multiplication

In this example $k = [\log_2 (M-1)] = 10$, so the algorithm will take at most 10 steps. The results of applying the iterative steps of the algorithm are shown in Table 1. The parity d_i of a_{i-1} is determined using Theorem 4 which requires the evaluation of the core of a_i and the parity of its residue components. The bit ε_i is determined by means of Theorem 7 which requires two core calculations and a comparison. In Table 1 six iterations are shown since $a_7 = 0$. The last entry for the c_i column gives c = (0,3,1,1).

TABLE 1

i	d_i	a_i	b_i	ε_i	c_i
0	-	(0,1,5,8)	(2,4,6,9)	0	(0,0,0,0)
1	0	(0,3,6,4)	(1,3,5,7)	1	(0,3,6,4)
2	0	(0,4,3,2)	(2,1,3,3)	1	(0,2,2,6)
3	0	(0,2,5,1)	(1,2,6,6)	0	(0,2,2,6)
4	0	(0,1,6,6)	(2,4,5,1)	1	(0,3,1,1)
5	0	(0,3,3,3)	(1,3,3,2)	0	(0,3,1,1)
6	1	(1,1,1,1)	(2,1,6,4)	0	(0,3,1,1)
7	-	0	–	–	–

(iii) Delta Multiplication

Since $M = 1155$, the subsystems the complete system is subdivided into are {3,11} with range $M_1 = 33$, and {5,7} with range $M_2 = 35$. For extension of base calculations core functions are defined on each of the two subsystems: $C_1(a) = [a/3] - [a/11]$, and $C_2(a) = -[a/5] + 3[a/7]$.

Since $a = (0,1,5,8)$ it follows that $r(a) = (0,8)$ in the first subsystem. Using the extension of base theorem and $C_1(r(a)) = 8$, one obtains that $r(a) = (0,0,2,8)$ in the complete system. The value of a^* is first evaluated modulo M_2 in the second subsystem,

$$a^* = \left| \frac{a - r(a)}{M_1} \right|_{M_2} = \frac{(1,5) - (0,2)}{(3.5)} = (1,3) \, x \, (2,3) = (2,2),$$

and then since $C_2(a^*) = 0$, by extension of base, obtain $a^* = (2,2,2,2)$.

Similar calculations lead to $r(b) = (1,4,6,1)$, $b^* = (2,1,5,4)$, $\Delta(a) = (2,3,2,1)$ and $\Delta(b) = (1,1,1,1)$. From these c is evaluated as $c = (2,2,2,2) \times (2,1,5,4) + (2,3,2,1) + (1,1,1,1) = (1,1,6,10)$.

The first two algorithms evaluate $c = (0,3,1,1)$, which corresponds to the integer 78. In fact since $a = 96$ and $b = 944$, the exact value of *[c] is 78*. Delta multiplication evaluates $c = (1,1,6,10)$ which corresponds to 76, i.e., an error of 2 units.

The core algorithm is exact but it is the most intensive in computations. The delta algorithm requires eight extensions of base, independently of the size of the moduli set, and assures a maximum error of less than 3. The binary algorithm is in general the simplest and of faster convergence, but allows the largest error, e.g., if in the same RNS one considers the case where $a = 1023$ and $b = 1154$, then $c = [a.b / M] = 1022$, but the binary algorithm calcualtes $c = 1013$, i.e., an error of 9.

Division Algorithm. The binary division algorithm for floating point arithmetic is similar to that for binary multiplication. It combines a binary expansion of a number in [0,1) with the quotients and remainders obtained in dividing the range of the RNS by successive powers of two.

Let a and b be two integers in $[0,M]$, $a < b$, which represent the mantissas a/M and b/M respectively. The quotient mantissa $(q' / M) = (a / M) / (b / M)$ is represented by q in $[0,M)$, where $q = [M.a/b]$.

The value of q' can be evaluated as the sum of the terms:

$$Q_1 = \sum_{i=1}^{k} \varepsilon_i . T_i ,$$

$$Q^*_2 = \sum_{i=1}^{k} (\varepsilon_i / 2^i).S_i ,$$

$$E_i = M. \sum_{i>k} \varepsilon_i / 2^i ,$$

where the ε_i are the bits in the binary expansion of a/b, i.e.,

$$(a/b) = \sum_{i=1}^{\infty} \varepsilon_i / 2^i ,$$

$k = [\log_2 M]$; and T_i and S_i are the quotients and remainders obtained when dividing M by successive powers of 2, i.e., $T_i = [M / 2^i]$, $S_i = |M|_{2^i}$.

The bits ε_i, as shown below, can be evaluated within the RNS, by successive multiplications by 2 and comparisons with b. The sum Q_1 only involves the bits and the system constants T_i, and so it is also computable in residue arithmetic. Applying this direct approach to the sum Q^*_2 proves not useful since $S_i < 2^i$ and hence the sum would be computed as $Q^*_2 = 0$. A more accurate procedure is obtained by first evaluating this sum with $M.S_i$ instead of S_i, and then dividing the result by M, i.e., Q^*_2 is rewritten as the sum of

$$Q_2 = \left[\frac{1}{M} \sum_{i=1}^{k} \varepsilon_i U_i \right]$$

$$E_2 = \frac{1}{M} \sum_{i=1}^{k} e_i V_i / 2^i ,$$

$$E_3 = \frac{1}{M} \sum_{i=1}^{k} \varepsilon_i . U_i , - Q_2$$

where $U_i = [M.S_i / 2^i]$, and $V_i = |M.S_i|_{2^i}$.

The value of $q = [a.b / M]$ is evaluated as the sum $Q_1 + [Q_2]$, where $E1$, $E2$, and $E3$ contribute to define the error in the evaluation of q. This error can be shown not to exceed

$1 + (M / 2^k) + (k-1 + 2^{-k}) / M < 4.$

The algorithm presented by Akushskii et al. can be divided into three parts.

(1) Evaluation of the system constants (precomputed)

$k = [\log_2 M],$

$T_i = [M / 2^i],\ i = 1,\dots,k,$

$U_i = [M. |M|_{2^i} / 2^i],\ i = 1,\dots,k,$

$C(U_i) =$ core of U_i.

(2) Evaluation of the binary expansion of a/b. The following recursive scheme is used: starting with initial values of $a_0 = a$ and $\varepsilon_0 = 0$, the recursive step is

$a_i = 2(a_{i-1} - b.\varepsilon_{i-1}),$

$$\varepsilon_i = \begin{cases} 1 & \text{if } a_i > b, \\ 0 & \text{otherwise.} \end{cases}$$

(3) Evaluation of the sums used to evaluate the quotient q.

$$Q_1 = \sum_{i=1}^{k} T_i \varepsilon_i$$

$$Q_2 = \left\lfloor \sum_{i=1}^{k} U_i \varepsilon_i / M \right\rfloor$$

$$q = Q_1 + Q_2.$$

Example 10. Consider the RNS with moduli {7,9,11}, and core function with weights {-1,-1,3}. Let a = (3,6,10)with $C(a)$ = 0, and b = (0,2,8) with $C(b)$ = 5, represent two mantissas.

The residue representation of T_i, U_i, and their cores are shown in Table 2.The iterations of step (2), are shown in Table 3. Each column corresponds to an iteration starting with the column of initial values. The row entries are evaluated as follows

(i)
$$a_i = \begin{cases} 2.a_{i-1} & \text{if } \varepsilon_{i-1} = 0 \\ 2.(a_{i-1}-b) & \text{if } \varepsilon_{i-1} = 1 \end{cases}$$

(ii) $C(a_i)$ evaluated using Theorem 2 for the core of $2.a_i$ or $2.(a_i-b)$, depending on the value of ε_i,

(iii) a_i-b computed in residue arithmetic

(iv) $C(a_i$-$b)$ computed using Theorem 2 for the core of a difference,

(v) $C(a_i$-$b)$ computed using Theorem 13, the Core Chinese Remainder Theorem,

(vi) binary bit: ε_i = 1 or 0 depending on whether the two core values given in (iv) and (v) coincide or not,

(vii) $Q_{1,i} = Q_{1,i-1} + T_i \cdot \varepsilon_i$

(viii) $MQ_{2,i} = MQ_{2,i-1} + U_i \varepsilon_i \pmod{M}$,

(ix) overflow indicator: η_i = 1 or 0 depending on whether the sum in (viii) overflows the system or not,

(x) accumulated overflow indicator: $sum\eta_i = sum\eta_{i-1} + \eta_i$,

(xi) $q_i = Q_{1,i} + sum\eta_i$

TABLE 2

	1	2	3	4	5	6	7	8	9
T_i	3,4,5	5,2,8	2,5,9	1,7,10	0,3,10	3,1,10	5,5,5	2,2,2	1,1,1
U_i	3,4,5	5,2,8	6,1,4	6,0,7	6,4,3	6,6,1	6,7,0	6,3,5	6,1,2
$C(U_i)$	6	2	8	3	9	12	7	9	5

TABLE 3

i	0	1	2	3	4	5	6	7	8	9
a_i	3,6,10	6,3,9	5,6,7	3,3,3	6,2,1	5,0,8	3,5,0	6,6,6	5,3,1	3,6,2
$C(a_i)$	0	2	6	13	13	9	8	0	1	1
a_i-b	--	6,1,1	5,4,10	3,1,6	6,0,4	5,7,0	3,3,3	6,4,9	5,1,4	3,4,5
$C(a_i$-b)	--	-3	-2	5	5	5	0	-8	-7	-7
$C(a_i$-b)	--	10	11	5	5	5	0	5	6	6
ε_i	0	0	0	1	1	1	1	0	0	0
Q_{1i}	0	0	0	2,5,9	3,3,8	3,6,7	6,7,6	6,7,6	6,7,6	6,7,6
Q_{2i}	0	0	0	6,1,4	5,1,0	4,5,3	3,2,4	3,2,4	3,2,4	3,2,4
η_i	--	0	0	0	0	1	1	0	0	0
Sum η_i	--	0	0	0	0	1	2	2	2	2
q_i	--	0	0	2,5,9	3,3,8	4,7,8	1,0,8	1,,0,8	1,0,8	1,0,8

REFERENCES

[1] I. Ja. Akushskii, V. M. Burcev, and I. T. Pak, "A new positional characteristic of nonpositional codes and its applications," in *Coding Theory and the Optimization of Complex Systems* (Ed. by V. M. Amerbaev; 'Nauka' Kazakh. SSR, Alma-Ata; 1977)

[2] I. Ja. Akushskii, V. M Burcev, and I. T. Pak, "Computation of the positional characteristic of a nonpositional code," in *Coding Theory and the Optimization of Complex Systems* (Ed. by V. M. Amerbaev; 'Nauka' Kazakh. SSR, Alma-Ata; 1977)

[3] D. D. Miller, J. N. Polky, and J. R. King, "A survey of recent Soviet developments in residue number systems," *Proceedings of the 26th Midwest Symposium on Circuits and Systems*, Puebla, Mexico, August 1983.

[4] N. S. Szabo', and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw Hill, 1967.

[5] R. S. Garfinkel, and G. L. Nemhauser, *Integer Programming*. New York: John Wiley & Sons, 1972.

[6] H. Greenberg, *Integer Programming*. New York: Academic Press, 1971.

[7] I. Ja. Akushskii, V. M Burcev, and I. T. Pak, "Algorithms for division using the core characteristic," in *Coding Theory and the Optimization of Complex Systems* (Ed. by V. M. Amerbaev; 'Nauka' Kazakh. SSR, Alma-Ata; 1977).

8] I. Ja. Akushskii, V. M Burcev, and I. T. Pak, "Algorithms for multiplication using a kernel characteristic," in *Theory of Coding and Complexity of Calculations* (Ed. by I. Ja. Akushskii, V. M Burcev, and I. T. Pak; 'Navka' Kazah. SSR, Alma-Ata; 1980).

9] I. Ja. Akushskii, V. M Burcev, and I. T. Pak, "Principles for the construction of highly efficient and reliable processors in non-positional number systems," in *Theory of Coding and Complexity of Calculations* (Ed. by I. Ja. Akushskii, V. M Burcev, and I. T. Pak; 'Navka' Kazah. SSR, Alma-Ata; 1980).

10] I. Ja. Akushskii, and V. M Burcev, "Some properties of the core characteristic of nonpositional systems," *Vestnik Akad. Nauk Kazakh. SSR*, 1983.

11] R. T. Gregory, and E. V. Krishnamurthy, *Methods and Applications of Error-Free Computation*, New York: Springer-Verlag, 1984.

12] C. Chiang and L. Johnsson, "Residue Arithmetic and VLSI", *Proc. IEEE Inter. Conf. on Computer Design: VLSI in Computers*, Port Chester, N. Y., 1983.

13] C. Yeh, I S Reed and J. J. Chang, "VLSI Design of Number-Theoretic Transforms for a Fast Convolution", *Proc. IEEE Inter. Conf. on Computer Design: VLSI in Computers*, Port Chester, N. Y., 1983.

14] G. A. Jullien and A. Bayoumi, "RNS Modules for VLSI Implementation of Digital Filters:, *Proc. 26th Midwest Symp. Circuits and Systems*, Puebla Mexico, August 1983.

15] W. K. Jenkins, "Residue Number System Error Checking Using Expanded Projection", *Electronics Letters*, Vol. 18, No. 21, pp. 927-928, Oct. 1982.

] F. J. Taylor, "A VLSI Residue Arithmetic Multiplier", *IEEE Trans. Computers*, Vol. C-31, No. 6. pp. 540-546, June 1982.

'] C. H. Huang, "High-Speed Two-Dimensional Filtering Using Residue Arithmetic" *24th SPIE Intern. Tech. Symp.*, San Diego, CA. July 1980.

ı] C. H. Huang, D. G. Peterson, H. E. Rauch, J. W. Teague, and D. F. Fraser, "Implementation of a Fast Digital Processor Using Residue Number Arithmetic", *IEEE Trans. Circuits and Systems*, Vol. CAS-28, No. 1, pp. 32-38, Jan. 1981.

)] W. K. Jenkins and B. J. Leon, "The Use of Residue Number Systems in the Deisgn of Finite Impulse Response Digital Filters", *IEEE Trans. Circuits and Systems*, Vol. CAS-24, No. 4, pp. 191-201, April 1977.

)] G. A. Jullien and W. C. Miller, "A Hardware Realization of an NTT Convolver Using ROM Arrays", *Proc. 1980 IEEE Intern. Conf. Acoustics Speech and Signal Processing*, Denver, CO, April 1980.

1] J. N. Polky and D. D. Miller, "The RAAF processor: architecture of the residue arithmetic adaptive filter processor", *Conference Record. 16th Asilomar Conference on Circuits, Systems & Computers*, Pacific Grove, CA., Nov. 1982.

2] M. A. Soderstrand, "New Hardware for High Speed Adaptive Digital Filtering", *Proc. 24th Midwest Symp. Circuits and Systems*, Albuquerque, NM. pp. 63-68, June 1981.